2



# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



# THESIS

A Prototype for Converting
Linear Programming (LP) Models
to
Structured Modeling Graphs

by

David Steven Hill

March 1989

Thesis Advisor: Daniel R. Dolk

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED | 1b RESTRICTIVE MARKINGS |
| 2a SECURITY CLASSIFICATION AUTHORITY | 3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S) | 5 MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable) | 7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School |
| 6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000 | 7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000 |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable) 54 | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c ADDRESS (City, State, and ZIP Code) | 10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO |

11 TITLE (Include Security Classification)
A Prototype for Converting Linear Programming (LP) Models to Structured Modeling Graphs

12 PERSONAL AUTHOR(S)
Hill, David S.

| 13a TYPE OF REPORT | 13b TIME COVERED | 14 DATE OF REPORT (Year, Month, Day) | 15 PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM ____ TO ____ | March 1989 | 145 |

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Modeling, Linear Programming, Model Management Systems, Structured Modeling, Mathematical Modeling |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Geoffrion's structured modeling provides a very promising framework for the development of future model management systems (MMS). This thesis presents a prototype that converts a mathematical representation of simple LP models to Geoffrion's structured modeling representations. The general procedures presented could be extended to convert an LP model represented in any precisely defined mathematical language. This would allow the development of integrated modeling environments based upon the structured modeling framework which would accept input in a number of common LP language formats.

| | |
|---|---|
| 20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS | 21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL Daniel R. Dolk | 22b TELEPHONE (Include Area Code) (408) 646-2260; 22c OFFICE SYMBOL 54DK |

DD FORM 1473, 84 MAR    83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE





UNCLASSIFIED

# A Prototype for Converting Linear Programming (LP) Models to Structured Modeling Graphs

By

David S. Hill
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1980

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March, 1989

Author: David Steven Hill

Approved By: Daniel R. Dolk, Thesis Advisor

Gordon H. Bradley, Second Reader

David R. Whipple, Chairman, Department of Administrative Sciences

Kneale T. Marshall,
Dean of Information and Policy Sciences

**ABSTRACT**

Geoffrion's structured modeling provides a very promising framework for the development of future model management systems(MMS). This thesis presents a prototype that converts a mathematical representation of simple LP models to Geoffrion's structured modeling representations. The general procedures presented could be extended to convert an LP model represented in any precisely defined mathematical language. This would allow the development of integrated modeling environments based upon the structured modeling framework which would accept input in a number of common LP language formats.

**TABLE OF CONTENTS**

# I. INTRODUCTION

The most widely accepted framework for building decision support systems (DSS) suggests three major components: the dialogue generation and management software which controls the DSS - user interface, the data base management software (DBMS) and the model management system software (MMS). [Ref. 1:p. 21]

Significant advances have been made in improving the dialogue management and DBMS components of DSS. Color graphics, windowing systems, pull down menus and simple input devices such as the mouse, provide the basic tools to build a user-friendly interface. The implementation of relational database theory has provided a number of powerful and flexible DBMS.

The third component, the MMS, is the area where the greatest amount of work remains to be done. Management science and operations research (MS/OR) models have made significant contributions in specific applications. However, these models have been largely stand-alone, costly to build, and have dealt primarily with well structured problem domains.

More recent modeling systems e.g., IFPS [Ref 2] have attempted to provide a more general tool for creating models

that assist the decision maker. The objective of these new systems is to increase the productivity of the model builder and to make the modeling process more acceptable to the non-technical user. Despite these improvements, no integrated modeling environment exists today that can meet the goals described by the Sprague and Carlson framework [Ref. 3:p 260].

This lack of progress is particularly troubling because the modeling component is the very heart of the DSS, As Sprague declares:

> ...it is the integration of models into the information system that moves an MIS which is based on integrated reporting and data base/ data communication into a full decision support system.[Ref. 1:p. 257]

Managers have traditionally been reluctant to recognize the value of MS/OR models. The reasons for this reluctance has been discussed at length in the literature [Ref. 3:p.259;Ref. 4:p. 466;Ref. 5:p. 36;Ref. 6:p. 704;Ref. 7:p.548]. The most common reason given for managers' lack of acceptance is the poor model/user interface in existing models. Many modeling software systems present the manager with unnecessary detail, are too technical in nature and are difficult for the manager to understand.

To overcome the managers' reluctance a MMS needs to combine the power of MS/OR algorithms for solving large, complex models with a flexible user interface that allows the

model and the results of the modeling process to be presented in a comprehensible manner to a manager. Geoffrion's structured modeling provides a formal framework for describing models which aims to provide the foundation for such a system [Ref. 7].

In this thesis we will construct a prototype parser which will convert mathematical representations of simple linear programming (LP) models to Geoffrion's structured modeling representations. We will attempt to demonstrate that the algorithms presented here can be extended to convert any LP modeling language to a structured modeling representation. This would allow development of integrated modeling environments based upon the foundation of structured modeling which could accept input in a number of common LP language formats.

This work is organized as follows: Sections II and III provide an overview of model management and structured modeling. Section IV discusses the algorithms for automatic generation of structured modeling representations from the LP language. Section V will present the implementation of the prototype parser and Section VI will present the limitations of the prototype as well as possible extensions for future development.

## II. MODEL MANAGEMENT

The rapid growth in the number of personal computers in recent years has fueled an interest in model-based decision making. The introduction of spreadsheet software has given the non-technical manager a user-friendly vehicle for creating models. These spreadsheets make it possible for the manager to create models for a wide variety of applications. (e.g., capital budgeting, human resource planning, resource allocation or portfolio selection)

While this growth in the number of models can have a positive effect on an organization it also creates a number of managerial problems. When important decisions are based upon models it is imperative that the models are valid, correctly applied and based upon current data. Serious questions exist about the validity, integrity and security of the spreadsheet models used in decision making. These problems are very similar to the problems that led to the realization of the need for effective data management.[Ref 5:p. 38]

The decentralized nature of spreadsheet modeling makes control very difficult. As the use of models continues to grow, it is important for managers to realize the potential

threat these problems pose for their organization. Managers must begin to recognize that models, like data, are an organizational resource that require management.

Model management systems (MMS) have been proposed to deal with these problems associated with decentralized modeling. An MMS performs functions for models analogous to those that a DBMS performs for data. An MMS contains a validated, well-documented model base accessible by all authorized users. The MMS must provide support for:

1. A consistent method for generating and updating models.[Ref 1:p. 262]
2. A flexible method for communicating modeling results in a manner suitable for technical and non-technical users.[Ref 7:p. 549]
3. Integration of models with the existing data.
4. Integration with advanced solver techniques.
5. A control mechanism for ensuring the security and integrity of models in the model base.[Ref 6]

The central MMS design issue is the method for representation and storage of the models in the model base. The model representation must be flexible enough to support the needs of all the users of the system. This requires a model representation that allows "views" of the model at different levels of complexity, including a analytical view

for the technical model builder and a natural language view for communication with the non-technical user of the MMS [Ref. 7:p. 549].

There have been a number of methods suggested for representing and storing the models in the model base.[Ref 1: p. 268] The traditional method is to represent models as subroutines in a high level language. In this approach the model base consists of a library of subroutines that is accessed by a subroutine call.

This is the "black box" method of modeling that has made managers reluctant to use models for decision making. The interaction between the model and the user is very limited. The user supplies the data and the model produces "the result". There is no feature for explaining the models or the assumptions upon which they are based.

A related approach represents models as statements in a modeling language such as GAMS.[Ref 8] In this approach the user defines the models in an algebraic language. The models are solved through the use of a common optimizer. This approach allows the user to focus on the modeling process rather than on developing the solver algorithm. Despite this improvement, the models-as-statements approach is limited in its ability to interact with the users. There is no feature for ad hoc queries of the model base. The algebraic

representation of the model is an improvement over the model-as-subroutines approach but is still inadequate for communication purposes.

The most promising approach represents models as data.[Ref 9:p. 36] This approach uses existing DBMS technology to store and access models, simplifying the integration of models and data and allowing flexible queries of the model base to aid the user throughout the modeling process.

Geoffrion's structured modeling offers a promising theoretical framework for implementation of a model management system. The structured modeling representation provides support for multiple views of models and graphical representations of models to enhance communication and improve acceptance of modeling by non-technical users.

Our prototype will use the models-as-data approach to store Geoffrion's structured modeling representation. Structured modeling is described in the next section.

# III. STRUCTURED MODELING

Structured modeling, developed by Geoffrion[Ref. 7], is a very general approach to modeling. Its goal is to foster development of a new generation of modeling systems with the following features [Ref. 7:p. 549]:

1. A conceptual framework for modeling based upon a single model representation format suitable for managerial communication, mathematical use and direct computer execution.
2. Independence of model representation and model solution
3. Sufficient generality to encompass most of the modeling paradigms that MS/OR and kindred model-based fields have developed.
4. Support for the entire modeling life cycle.
5. Integrated facilities for data management and ad hoc queries.
6. Desktop implementation with a modern user's interface,including immediate expression evaluation as in spreadsheet software.

Our prototype will convert mathematical representations of LP models to Geoffrion's structured modeling representations. Here we provide the basics of structured modeling that are relevant to our prototype and discuss the features of structured modeling that make it attractive as a basis for MMS development.

We will provide an example of how a LP model is represented in structured modeling. The example model we have

chosen is the same classic transportation problem used by Geoffrion [Ref. 7: p. 570]. This model describes plants which manufacture a product that must be shipped to customers. There are production constraints for each plant and demand constraints for each customer. The objective is to minimize the total cost of shipping the product within the constraints given.

We will also use this example in Section V, which will allow the reader to compare the representation here to the one generated by our prototype. This informal approach provides only those elements of structured modeling necessary to understand the prototype implementation. For a more complete coverage of the subject see Geoffrion [Ref. 7].

## A. PRINCIPLES OF STRUCTURED MODELING

A structured model is composed of elements. These elements are either primitive or else defined in terms of their relationship to the other elements. There are six types of elements:

1. **Primitive entity elements** (pe) have no value and usually represent things or concepts in the model (e.g., a plant in the transportation problem).

2. **Compound entity elements** (ce) have no value and usually represent concepts that are defined in terms of other things or concepts. (e.g., a plant-customer link in the transportation problem defined in terms of a certain plant and a certain customer).

3. **Attribute elements** (a) have a constant value and represent a property of a thing or concept (e.g., the supply capacity of a particular plant in the transportation problem).

4. **Variable attribute elements** (va) are like attribute elements but their value is discretionary and likely to change. (e.g., the flow of goods over a particular plant-customer link in the transportation problem).

5. **Function elements** (f) have a value that is derived by a specific equation or rule (e.g., the total cost associated with all flows in a transportation problem).

6. **Test elements** (t) are like function elements except that their value must be either true or false. (e.g., whether the demand requirement is met for a particular customer in the transportation problem).

The structured modeling framework consists of three levels: elemental structure, generic structure, and modular structure.

The elemental structure is the most basic level of the model. It provides the details of a specific instance of the model. Geoffrion defines the elemental structure in terms of a directed acyclic attribute graph of elements (nodes) and calls[1] (directed arcs). In all but the simplest of model instances the elemental graph will contain many arcs and nodes. In general this graph is too cluttered to be useful.

The generic structure is a generalization of the elemental structure. This structure captures the natural familial groupings of elements. Similar elements are grouped such that every element in a genus calls the same genera and is called

[1]A "call" represents the participation of the called element in the definition of the calling element. The head node is the calling element and the tail node is the called element.

called by the same genera. This property is called *generic similarity*. [Ref. 7:p. 553]

The generic structure can also be represented by a directed acyclic attribute graph called a genus graph. Figure 1 is an example genus graph for the transportation problem.

The genus graph is an example of the communication value of the structured modeling representations. The genus graph is dimension independent [Ref 7:p. 556] thus providing an insight into how the model works without the unnecessary details of the elemental structure.

The modular structure groups elements into conceptual units called modules. For example, in the transportation problem, the customer genus and the demand genus could be grouped into a "customer data" module. These modules allow the model to be viewed at different levels of complexity. The modular structure can be represented graphically as a rooted tree. The root represents the entire model and each terminal node is a genus. Only certain modular structures are allowed. Valid structures can be represented by an indented list that contains no forward references. This indented list is called the modular outline.

In addition to the graph based representations of models, Geoffrion has proposed a structured modeling language (SML) for representing schemas to reflect the generic and modular structure of models.[Ref. 10] Figure 2 is an example of the schema for the transportation problem.

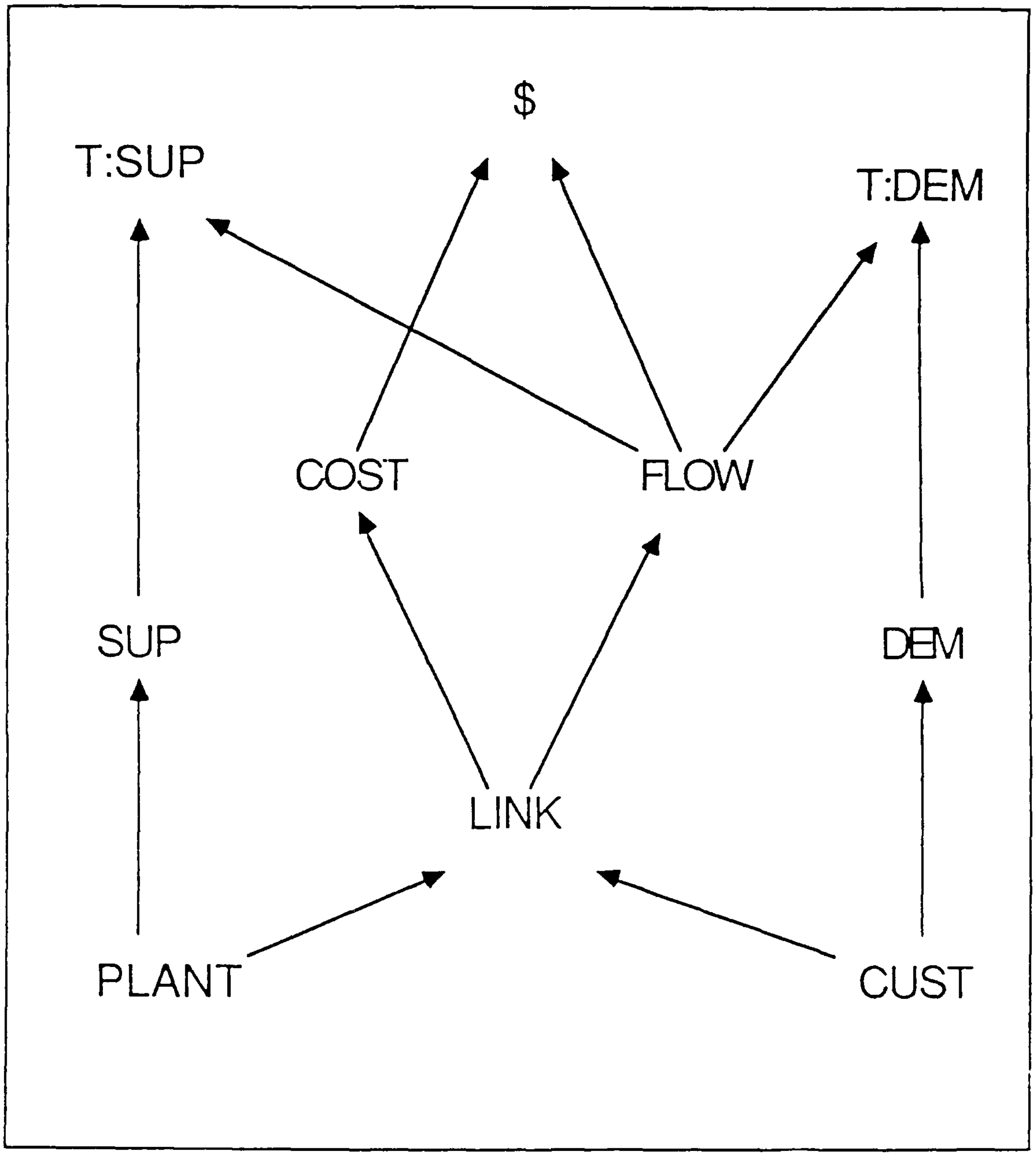

Figure 1: Genus Graph for the Transportation Problem

**&SDATA** SOURCE DATA

**PLANTi /pe/** There is a list of PLANTS.

**SUP(PLANTi) /a/** {PLANT}:R+ Every PLANT has a SUPPLY capacity measured in tons.

**&CDATA** CUSTOMER DATA

**CUSTj /pe/** There is a list of CUSTOMERS.

**DEM(CUSTj) /a/** {CUST} :R+ Every CUSTOMER has a nonnegative DEMAND measured in tons.

**&TDATA** TRANSPORTATION DATA

**LINK(PLANTi,CUSTj) /ce/** Select {PLANT}X{CUST} where i covers {PLANT},j covers {CUST} There are some transportation LINKS from PLANT to CUSTOMERS. There must be at least one LINK incident to each PLANT, and at least one LINK incident to each CUSTOMER.

**FLOW(LINKij) /va/** {LINK} :R+ There can be a nonnegative transportation FLOW (in tons) over each link.

**COST(LINKij) /a/** {LINK} :R Every LINK has a TRANSPORTATION COST RATE for use in $/ton.

**$(COSTij,FLOWij) /f/**;SUMiSUMj(COSTij*FLOWij) There is a TOTAL COST associated with all flows.

**T:SUP(FLOWij,SUPi) /t/** {PLANT} ;SUMj(FLOWij) <= SUPi Is the total FLOW leaving the PLANT less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

**T:DEM(FLOWij,DEMj) /t/** {CUST} ;SUMi(FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

Figure 2: Schema for the Transportation Model[Ref. 7:p. 570]

This SML-based schema defines the entire model structure independent of the elemental detail and is precisely defined to allow direct computer execution [Ref. 7:p. 562], [Ref. 10]. We will only provide an overview of the schema syntax here.

The schema is composed of two kinds of paragraphs: module paragraphs and genus paragraphs. The paragraphs are indented and organized in the same monotone order as the modular outline. Each paragraph consists of a formal part followed by an optional informal text interpretation part.

A module paragraph consists of the mnemonic module name, preceded by an ampersand (&) and an optional interpretation. Geoffrion argues <u>strongly</u> that the interpretation is a critical part of the modeling process.

The genus paragraph syntax will vary by element type. Figure 3 gives the general syntax for a genus paragraph. Optional items are enclosed in brackets.

```
GNAME [new index][(generic calling sequence)]
/type/ [index set statement][:range statement]
 [;generic rule statement] [interpretation].
```

Figure 3: General Syntax for a Genus Paragraph

**GNAME** is the mnemonic genus name. The genus name is followed by an index for those genera that are self-indexed.

**/type/** is the genus type declaration and must correspond to one of the six element types defined in structured modeling. The **index set statement** defines the permissible population of the genus. The **range statement** defines the permissible values for an attribute or variable attribute genus. The **generic rule** defines the rule by which the values of a function or test genus are derived.

The structured modeling framework has much to offer as a foundation for future model management systems. In a single model representation it provides a computer executable model definition <u>and</u> a flexible communication device.

Our prototype will focus on the genus graph and the text based schema representations of structured modeling. We will demonstrate how the structure of a LP model determines its structured modeling representation. The next section presents the theory upon which our conversion algorithm is based.

## IV. GENERATION OF GRAPHS FROM LINEAR PROGRAMMING MODELS

A linear program (LP) typically deals with the problem of allocating limited resources, subject to a set of constraints, in a way that maximizes return or minimizes cost.[Ref 11:p. 156]

This section will describe our method of converting mathematical representations of simple[2] LP models to structured model representations. This algorithm is based upon the relationship between LP model components and structured model genus types identified by Geoffrion and further amplified by Dolk.[Ref 12]

Figure 4 is a the standard form of the LP model using our mathematical notation. The following conventions are used:

1. Summations are identified by the token @SUM. The expression following the summation will be enclosed in parenthesis. (e.g., @SUMi(Xi) or @SUMiSUMj(Xij) )
2. All variables and coefficients are in uppercase.
3. All indices are in lower case.

Appendix A contains a precise definition of our notation. This mathematical notation is a su. 't of the generic rule grammar defined by Geffrion.[Ref 10:p. A4-1]

[2] Simple in this context means that there are no indices that depend upon other indices and that all indices range over their full set of values.

```
maximize:    (0)  Z = @SUMj(Cj * Xj)

subject to:  (1)  @SUMj(Aij * Xj) <= Bi

             (2)  Xj >= 0
```

Figure 4: Standard mathematical form of LP model

This model consists of an objective function (equation (0)), a set of constraints (equations (1) and (2)), coefficients (C and A),right hand sides (0 and B), decision variables (X), and indices (i and j).

Dolk has identified the following relationships between the components of an LP model and structured modeling genus types [Ref 12:p. 3]:

1. Each index is a primitive entity.
2. The objective function is a function genus.
3. Each constraint corresponds to a test genus.
4. Coefficients are attributes associated with the primitive entity or compound entity that corresponds to its index.
5. Right hand sides are attributes.
6. Decision variables are variable attributes.
7. For components with multiple indices (e.g., Aij) the multiple indices correspond to a compound entity.

The next requirement is to define the calling sequence which determines the genus graph structure. Dolk provides the following propositions [Ref 12:p. 3]:

1. Every test genus is a leaf node[3] in the genus graph.

2. The objective function of the mathematical model is a leaf node in the genus graph.

3. Each index in the mathematical model is a root node.[4]

4. The function and test genera will have only attributes or variable attributes in their calling sequences.

5. All variable attributes will appear in at least one test genus' calling sequence and the objective function genus.

6. Any component with multiple indices will have several primitive entities in its calling sequence, one for each index.

From these propositions we can identify each genus, its genus type, and its calling sequence. Further, each equation represents the generic rule for the corresponding test or function genus.

With this information we can construct the genus graph from a mathematical representation of a model. Figure 5 is an example genus graph constructed for the standard form of the LP model. This graph, while accurate, is not particularly enlightening without meaningful mnemonic names to identify each genus node. Our parser, described in detail in the next section, will allow the user to add mnemonic names.

---

[3] A leaf node appears in no other genus' calling sequence. In the graph it will have no outgoing arcs.

[4] A root node has no calling sequence. It will have no incoming arcs in the graph.

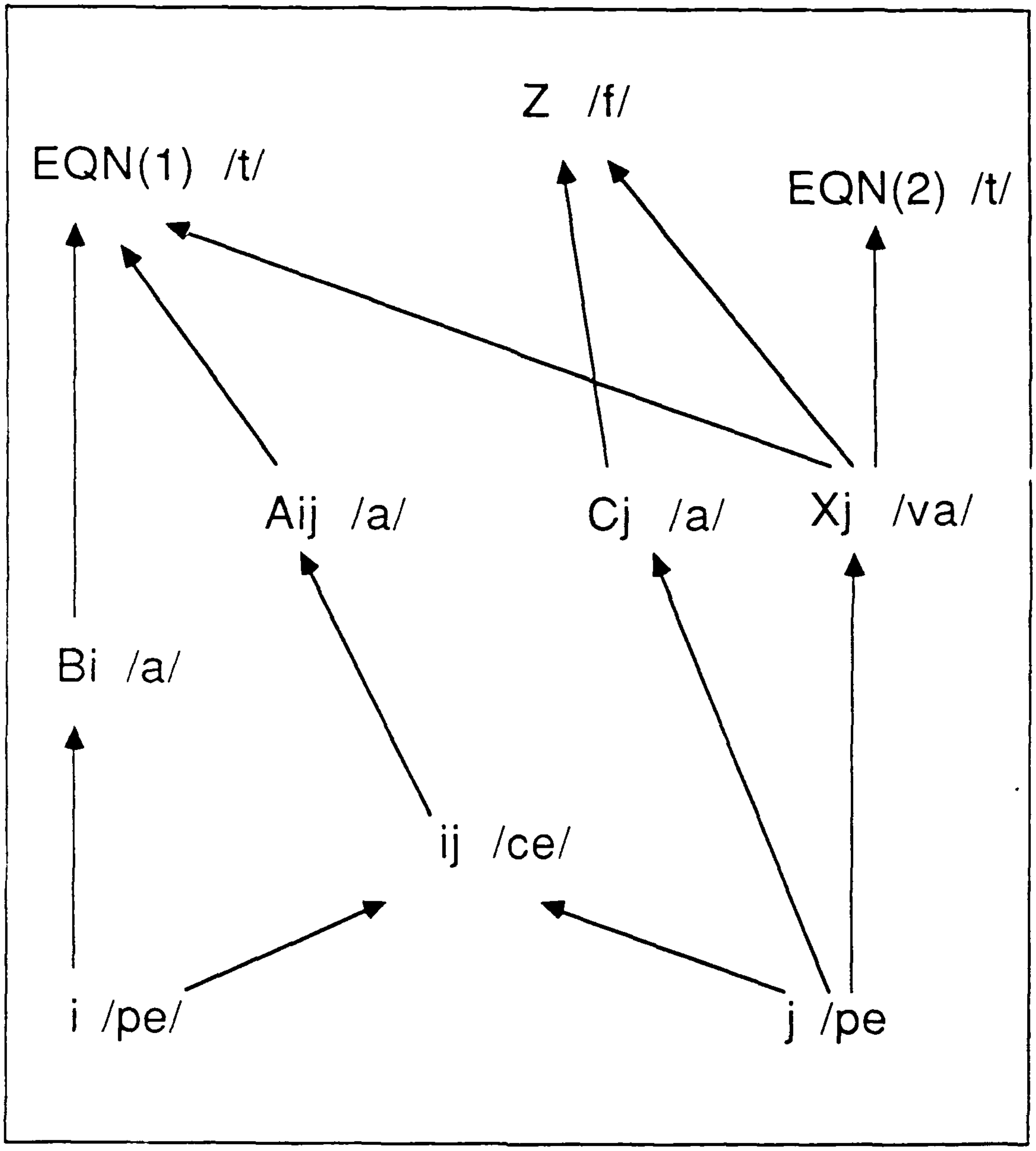


Figure 5: Genus Graph for the Standard Form of the LP model

We cannot determine the generic range of attribute genera, the index set statement, or the natural language interpretation from the mathematical model. However, we can construct an approximation of the text based schema.

Our prototype will parse each equation and create a symbol table for the model. We then apply the relationships and propositions described here to construct the structured modeling representations from the symbol table. The next section will provide the implementation details of the prototype and provide an example conversion of the transportation model.

## V. IMPLEMENTATION OF THE PROTOTYPE

Our prototype, LP/SM, was designed to allow the user to enter a mathematical description of a simple LP model, convert it to a corresponding structured modeling representation, adding mnemonic names if desired, and then display this model graphically as a structured modeling genus graph or as an SML-based schema. An edit feature allows the user to enter the natural language interpretation, the generic range and the index set statement which cannot be automatically generated by LP/SM. Here we describe the LP/SM implementation and provide an example conversion of the transportation problem.

### A. HARDWARE USED IN THIS IMPLEMENTATION

LP/SM was designed on an IBM PS/2 model 80 running the MS-DOS operating system. We found this to be an excellent environment for development. The 80386 CPU, provides rapid response and sufficient computing power for expansion of the prototype. The Video Graphics Array (VGA) graphics capability of the PS/2 ensures compatibility with all current PC graphics standards. The extended memory of the PS/2 makes it possible to implement applications that are memory intensive[5] (e.g., graphics).

---

[5]Our prototype requires at least 1.5 MB of memory to support the ORACLE RBDMS.

LP/SM was designed to ensure portability to the IBM PC AT environment. This required using Enhanced Graphics Array (EGA) graphics for display of the genus graph reducing the screen resolution to 640 pixels horizontally by 350 pixels vertically (640x350). While this resolution is limiting and makes some of the genus graph arcs appear jagged, it was sufficient for our prototype implementation. Our recommendations for future graphics enhancements are discussed in Section VI.

**B. SOFTWARE USED IN THIS IMPLEMENTATION**

The parser for our prototype was generated using automatic program generators. The program generators we chose were LEX: A Lexical Analyzer Generator [Ref. 13] and YACC: Yet Another Compiler Compiler [Ref. 14]. We chose this approach for a number of reasons:

1. The resulting product is produced more quickly.
2. The resulting product is flexible and adaptable. YACC will "read" any input that can be can be defined in its specification language. This specification language closely resembles BNF notation.[6]

LEX and YACC were designed to work together to build a parser for processing the input to a computer program. The user provides LEX and YACC with a high-level description of

[6]YACC will accept a very general class of grammars - LALR(1) with disambiguity rules.[Ref 15:p. 1]

the input language and these tools generate the source code for a parser which is capable of recognizing legal constructs of the defined language. LEX and YACC were written for the UNIX operating system, however their output is C source code which can be ported to the MS-DOS environment.

The remainder of the prototype was written in Microsoft C. We found Microsoft C had a number of advantages for our prototype implementation:

1. Complete MS-DOS function library and ANSI standard library functions to ensure compatibility with the code generated by LEX and YACC.

2. Complete Graphics function library.

3. Compatibility with the code generated by the ORACLE precompiler.

4. Helpful programmer support tools including the **make** facility for managing maintenance of source code and a powerful source level debugger.

Our prototype uses the ORACLE relational database management system (RDBMS) to store the structured models. ORACLE was chosen because it fully supports the ANSI standard SQL data manipulation language and contains a high-level programming language interface.

SQL provides powerful data manipulation functions allowing for flexible queries of our model base and eventual integration of models with their corresponding data tables.

ORACLE's C language interface extends the data manipulation functions of SQL by allowing the use of procedural programming language constructs (e.g., IF-THEN ELSE statements).

## C. DESIGN OVERVIEW

LP/SM consists of a number of separate modules that correspond to the functions displayed on the main menu. This modular approach will simplify future enhancements to the prototype.

Listed below are the options that appear on the main menu when the program is run:

1. Enter Model
2. Edit Model Schema
3. Display Model Graph
4. Display Model Schema
5. Quit

This menu was designed to function with a multiple-model model base, however since our prototype only allows a single model the user must initially select the "Enter Model" option. Any other selection will result in a pop-up window containing an error message.

When the "Enter model" option is selected the prototype performs the following functions (See Figure 6):

1. Parses each equation of the mathematical representation of the model to ensure the syntax is correct. When a valid equation is recognized the equation tokens are entered into the parser's symbol table.

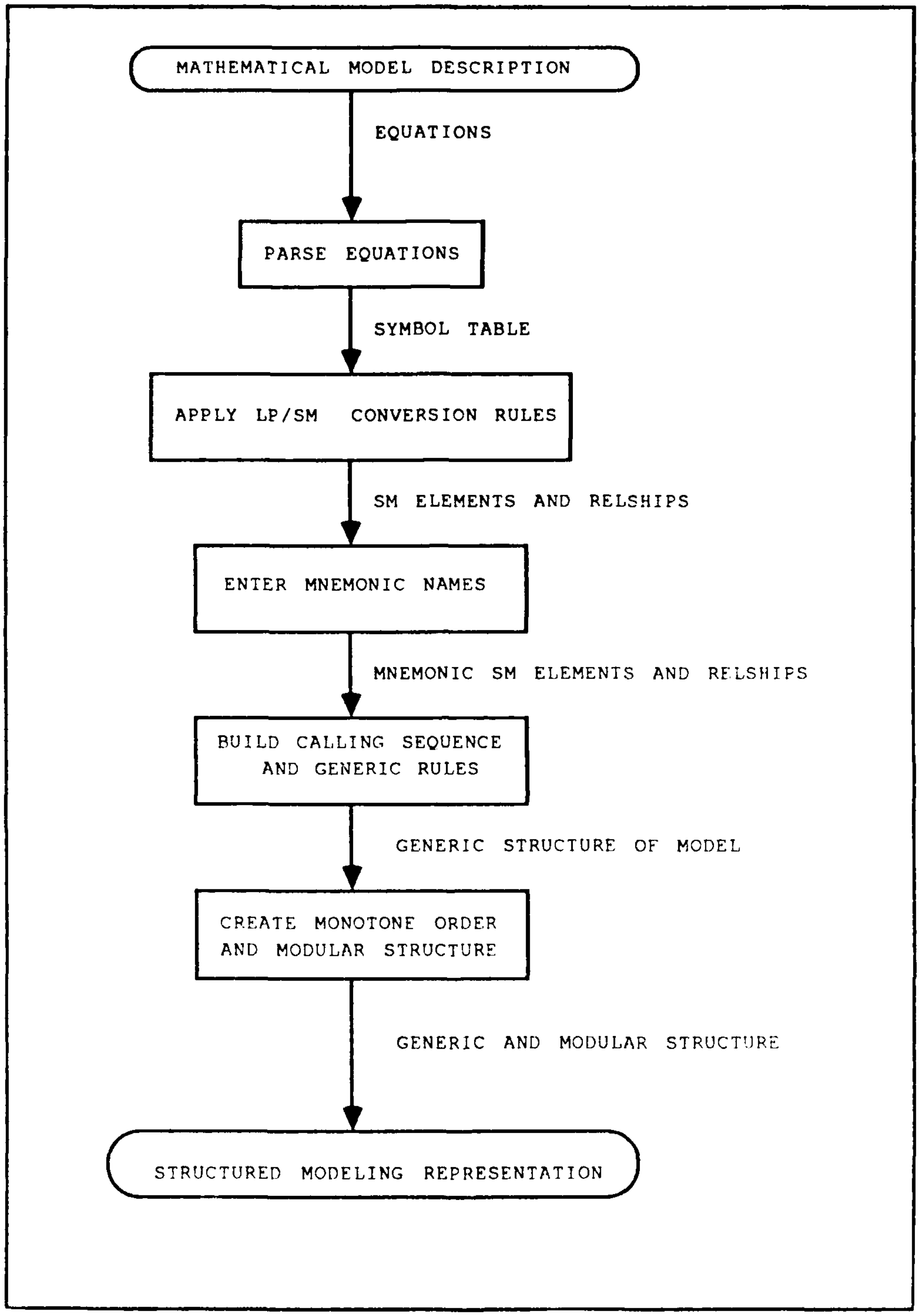


Figure 6: Overview of the LP/SM conversion process

2. Translates each element in the parser's symbol table entries to the corresponding structured modeling elements using the relationships and propositions described in Section IV.

3. Accepts the user defined mnemonic names corresponding to each structured modeling element found in the translation of the symbol table.

4. Substitutes user mnemonic names for each element, builds the mnemonic calling sequence for each of the non-primitive elements and constructs the mnemonic generic rule for each of the test and function elements.

5. Builds a simple modular outline where each element is a module of the model with no sub-modules. The monotone order is determined by a simple topological sort of the generic structure calls. This topological sort is modified to place the attributes that call primitive entities immediately following the primitive entity in the modular outline. This is done to improve the appearance of the model schema.

6. Writes the structured modeling description of the LP model to the ORACLE database.

The following sections describe the implementation of each of the main menu functions.

### 1. Parser Implementation

The first step in constructing a parser is to define the language to be parsed. To define any input language we must clearly specify the basic symbols allowed (tokens) in the language and the define rules for combining these symbols into legal constructs (non-terminal symbols) in the language. These tokens and the rules for combining them comprise the grammar for the input language.

The final element of language definition is the semantics of the language. Semantic rules determine if a language construct that meets the syntax requirements is

meaningful. For example if the syntax requires that an index follow the @SUM token, the semantic rules would determine if the index used after the @SUM token is correct in the context of the entire equation. Semantic rules are much more difficult to define. Our prototype parser deals solely with the syntax of the language. This limitation is discussed further in Section VI.

Use of the program generators requires defining the input language in two steps. The input to LEX defines the regular expressions that represent legal tokens of the language and actions to take when a token is found [Ref 13:p. 1]. The input to YACC defines the rules by which these tokens can be combined to form legal constructs of the language and the actions to take when a legal construct is recognized [Ref 14:p. 1]. Appendix B contains our input to LEX and YACC.

LEX produces a lexical scanner which reads from the input character stream and breaks it into legal tokens of the language. These tokens are passed to the parser created by YACC until a legal language construct is found or a syntax error occurs.

When the "Enter Model" option is selected, the user will be prompted to enter the model name. The user will then be prompted to enter the objective function. Each equation

will be parsed as it is entered. If a syntax error occurs an error message will appear and the user must reenter the objective function.

The cursor keys combined with the insert and delete keys provide a simple edit feature to assist the user in correcting the syntax error.

When an equation is correctly entered the user will be prompted for the next equation. This process is repeated for all the equations in the model. To complete the model definition the user must enter the token END when prompted for the next equation. Figure 7 is a description of the transportation model in our mathematical language. This is the form that would be entered in LP/SM.

```
@SUMi SUMj (Cij * Xij)
@SUMj (Xij) <= Si
@SUMi (Xij) = Dj
Xij >= 0
```

Figure 7: Mathematical Language Description of the Transportation Problem

After the END token is entered, the user will be allowed to enter meaningful mnemonic names for the model elements. The user must identify which element is the decision variable. This allows us to identify which of the

attribute elements are variable attributes. Figure 8 is a chart showing the mnemonic names describing the transportation model. After the last mnemonic name is entered the model description is completed, written to the ORACLE tables and the main menu is redisplayed.

| GENUS NAME | GENUS TYPE | MNEMONIC NAME |
|---|---|---|
| M_TRANS | model | M_TRANS |
| EQNO | f | TOTAL |
| EQN1 | t | T:SUP |
| EQN2 | t | T:DEM |
| EQN3 | t | NON_NEG |
| ij | ce | LINK |
| i | pe | PLANT |
| j | pe | CUST |
| C | a | COST |
| X | va | FLOW |
| S | a | SUP |
| D | a | DEM |

Figure 8: Mnemonic Description of the Transportation Problem

### 2. Storage of the Models in the Model Base

The structured model is represented in two ORACLE tables: **RELSHIP** and **ENTITY.** This structure was developed by Dolk [Ref. 6:p. 710] to represent a structured model as part of an information resource dictionary system (IRDS). This representation offers a number of advantages [Ref 6: p.718]:

1. Model integrity consistent with structured modeling principles can be checked automatically with a single DBMS query command.

2. A wide variety of queries is available for both pre- and post-solution model analysis using DBMS query commands.

3. Modeling and data resources are consolidate in a single, shared environment.

4. The IRDS can be activated to interface with external processes such as optimization algorithms to support model manipulation and eventually serve as the foundation for a fully functional model management system.

Figure 9 lists the SQL commands that create the tables and views representing a structured model in the model base. The **CALLS** view represents the model's generic structure. The **CONTAINZ** view represents the model's modular structure. An additional view is created for each of the element types to support queries concerning the model structure.

In the **ENTITY** table **ename** is the mnemonic name of the model element, **etype** is the element type and must correspond to one of the six structured modeling element types (pe, ce, a, va, t, f). **dname** is the descriptive name for the genus. The **date_added, last_mod** and **nmods** fields are used for control purposes to record the date the model element was added or modified and the number of modifications that have been made to the model element. The **idx**, **idx_stmt**, **grange**, and **grule** fields are used to store the elements index, index statement, generic range statement, and generic rule. The **comments** field corresponds to the structured modeling natural language interpretation.

```
create table entity
(ename              char(20)  NOT NULL,
 etype              char(12)  NOT NULL,
 dname              char(30),
 date_added         date,
 last_mod           date,
 nmods              number(5),
 idx                char(4),
 idx_stmt           char(100),
 grange             char(20),
 grule              char(100),
 comments           char(100)  );


create table relship
(rtype              char(10) NOT NULL,
 e1name             char(20) NOT NULL,
 e1type             char(12) NOT NULL,
 e2name             char(20) NOT NULL,
 e2type             char(12) NOT NULL,
 rel_pos            number(2));


create view calls as
select e1name,e1type,e2name,e2type
from relship
where rtype = 'CALLS';


create view containz as
select e1name,e1type,e2name,e2type,rel_pos
from relship
where rtype = 'CONTAINS';
```

Figure 9: SQL commands for Model representation

In the **RELSHIP** table the **rtype** is "CALLS" for rows representing calls in the generic structure or "CONTAINS" for rows representing module containment. **e1name** and **e1type** contain the name and type of the calling element in the generic structure or the module name in the modular structure. **e2name** and **e2type** contain the name and type of the called element in the generic structure or the name and type of the element that is contained within the module. **rel_pos** is an number representing the order of the element in the modular outline.

Figure 10 shows the tables that are created for the transportation model. The prototype creates a simple modular outline. The module name is preceded by "M_" rather than the ampersand suggested by Geoffrion because the ampersand is a reserved symbol with special meaning in ORACLE.

### 3. Display of the Model Schema

When the "Display Model Schema" option is selected on the main menu, LP/SM reads the **ENTITY** table from the model base. This is accomplished by the use of the SQL command in Figure 11. This command ensures the schema is displayed in monotone order.

Figure 12 is an example of the schema created for the transportation model. This is our initial approximation of the SML-based schema. It does not contain the generic

**Structured Modeling Elements**

```
ENTITY(ename,etype,.....idx,idx_stmt,grange,
                                         comments,grule)
(M_TRANS,model,...,"","","","","")
(TOTAL  ,f     ,...,"","","","",@SUMiSUMj(COSTij*FLOWij)
(T:SUP  ,t     ,...,"","","","",@SUMj(FLOWij) <= SUPi)
(T:DEM  ,t     ,...,"","","","",@SUMi(FLOWij) = DEMj)
(NON_NEG,t     ,...,"","","","",FLOWij >= 0)
(LINK   ,ce    ,...,"","","","","")
(PLANT  ,pe    ,...,i ,"","","","")
(CUST   ,pe    ,...,j ,"","","","")
(COST   ,a     ,...,"","","","","")
(FLOW   ,va    ,...,"","","","","")
(SUP    ,a     ,...,"","","","","")
(DEM    ,a     ,...,"","","","","")
```

**Generic Structure**

```
CALLS(e1name,e1type,e2name,e2type)
   (COST   ,a ,LINK  ,ce )
   (LINK   ,ce,PLANTi,pe )
   (LINK   ,ce,CUSTj ,pe )
   (TOTAL  ,f ,COST  ,a  )
   (FLOW   ,va, LINK ,ce )
   (TOTAL  ,f ,LINK  ,ce )
   (T:SUP  ,t ,FLOW  ,va )
   (SUP    ,a ,PLANTi,pe )
   (T:SUP  ,t ,SUP   ,a  )
   (T:DEM  ,t ,FLOW  ,va )
   (DEM    ,a ,CUSTj ,a  )
   (T:DEM  ,t ,DEM   ,a  )
   (NON_NEG,t ,FLOW  ,va )
```

**Modular Structure**

```
CONTAINZ(e1name,e1type,e2name,e2type,rel_pos)
      (M_TRANS,model,COST    ,a , 8)
      (M_TRANS,model,FLOW    ,va, 9)
      (M_TRANS,model,TOTAL   ,f ,11)
      (M_TRANS,model,NON_NEG,t ,14)
      (M_TRANS,model,PLANTi  ,pe, 1)
      (M_TRANS,model,SUP     ,a , 2)
      (M_TRANS,model,CUSTj   ,pe, 3)
      (M_TRANS,model,DEM     ,a , 4)
      (M_TRANS,model,LINK    ,ce, 5)
      (M_TRANS,model,T:SUP   ,t ,12)
      (M_TRANS,model,T:DEM   ,t ,13)
```

Figure 10: ORACLE tables created to represent the transportation problem

```
Select    ename,etype,dname,idx,idx_stmt,
          grange,grule,comments
from      ENTITY,CONTAINZ
where     ENTITY.ename = CONTAINZ.e2name
order     by rel_pos;
```

Figure 11: SQL command to read **ENTITY** table from the model base

range, index set statement, or natural language interpretation portions of the genus paragraphs. These portions can be added by using the edit feature on the main menu.

#### 4. Display of the Genus Graph

The algorithm for creating the graph was adapted from the work done by Wyant [Ref. 16]. Figure 13 is the genus graph created by our prototype for the transportation problem.

When the "Display Model Graph" option is selected on the main menu, LP/SM reads the **CALLS** view from the model base. Figure 14 is the SQL command that reads the **CALLS** view from the model base. Each **CALLS** row represents a directed arc in the genus graph. We order the rows in the **CALLS** view to group all the calls to a particular node. This provides a more orderly presentation of the arcs when we draw the graph on the screen.

The genus graph is represented internally by creating a linked list containing the positions of each node

```
PLANTi /pe/    .

SUPPLY (PLANTi) /a/    .

CUSTj /pe/     .

DEMAND (CUSTj) /a/     .

LINK (CUSTj,PLANTi) /ce/    .

COST (LINK) /a/    .

FLOW (LINK) /va/   .

TOTAL (COST,FLOW) /f/ ;@SUMiSUMj(COSTij*FLOWij) .

T:SUP (FLOW,SUPPLY) /t/ ;@SUMj(FLOWij) <= SUPi   .

T:DEM (DEMAND,FLOW) /t/ ;@SUMi(FLOWij) = DEMj   .

NON_NEG (FLOW) /t/ ; FLOWij >= 0  .
```

Figure 12: SML-based schema created for the transportation problem

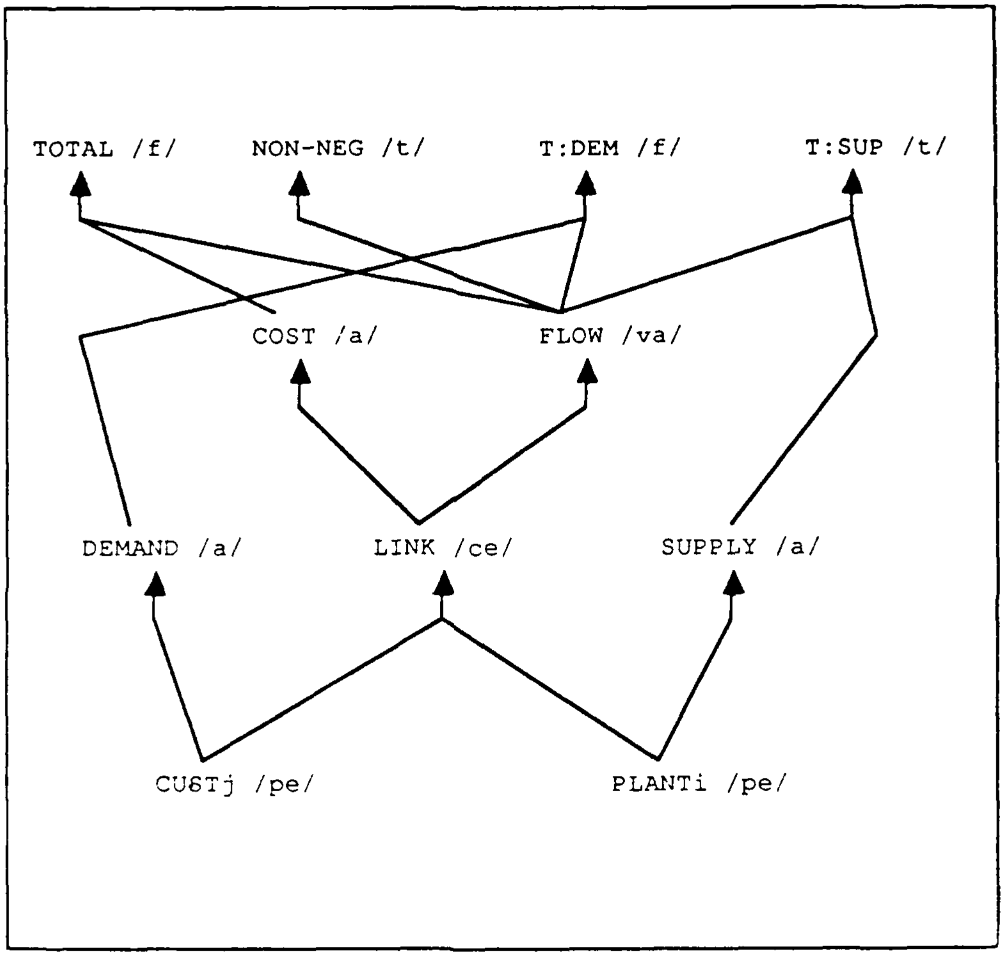


Figure 13: Genus graph generated for the transportation problem

```
Select e1name, e1type, e2name,e2type
from CALLS
order by e2name,e1name
```

Figure 14: SQL command to read CALLS view from the model base

in the graph. Each node in the linked list is a record containing the name and type of the node and the x and y screen coordinates of the center of the node.

The relationships defined by the **CALLS** view are used to draw the directed arcs between the nodes. The graph is centered on the screen. The x and y coordinates are computed by determining the number of levels on the graph and the number of nodes on each level.

Since the primitive entities represent root nodes in the genus structure, we add all the primitive entities to the position linked list on the first level of the graph. The next step is to search the **CALLS** view for all elements that call the primitive entities. These elements are added to the position linked list on level two. We then add the elements that call the elements on level two to the linked list on level three. This process continues until all genus graph nodes are represented in the position linked list.

It is possible that a genus will call genera that exist on two different levels of the graph. If this occurs we add a spaceholder position on the lower level to allow us to draw the directed arcs that span levels without drawing arcs that interfere with other nodes. For example in Figure 13 the arc from DEMAND to T:DEM uses a spaceholder on level two.

The most difficult issue in constructing the graph is how to place the nodes (elements) and directed arcs (calls). We have attempted to address the problem in our prototype. Wyant's "spaceholder" reduces the possibility that an arc will be drawn through a node and our ordering of the elements in the CALLS view improves the presentation of the arcs in the lower levels of the graph. While our prototype will always produce a graph that correctly represents the generic structure of the model, the graph's aesthetic quality is often lacking. This limitation will be discussed in Section VI.

**5. Editing of the Model Schema**

The "Edit Model Schema" option on the main menu allows the user to enter the components of the model schema that were not automatically generated by our prototype. This information is added to the ENTITY table of the model base and will be displayed in the model schema. Figure 15 shows the

model schema for the transportation problem after the index set statements, generic ranges and natural language interpretations have been added.

## D. CONSTRUCTION OF THE PROTOTYPE

The prototype is constructed in the following order:

1. Run LEX using the input listed in Appendix B.
2. Run YACC using the input listed in Appendix B.
3. Precompile all source modules containing ORACLE SQL commands with the PCC ORACLE precompiler.
4. Compile all source modules and link them into the executable file XMMS.EXE.

Appendix C contains the makefile used to compile the prototype and the C source code for all functions except those that were generated by YACC and LEX. The YACC and LEX functions were omitted because they can easily be reconstructed by using the input described in Appendix B.

## E. RUNNING THE PROTOTYPE

The prototype assumes that ORACLE is loaded into extended memory prior to running the prototype. ORACLE is loaded simply by typing ORACLE at the command prompt.

Once ORACLE is loaded the prototype is run by typing XMMS at the command prompt. This will clear the previous model (if one exists) from the ORACLE database and display the main menu.

```
PLANTi /pe/ There is a list of PLANTS.

SUPPLY (PLANTi) /a/ {PLANT} :R+  Every Plant has a
SUPPLY capacity measured in tons.

CUSTj /pe/  There is a list of CUSTOMERS.

DEMAND (CUSTj) /a/ {CUST} :R+ Every customer has a
nonnegative demand measured in tons.

LINK (CUSTj,PLANTi) /ce/  Select {PLANT}X{CUST}
where i covers {PLANT},j covers {CUST} There are
some transportation LINKS from PLANT to CUSTOMER.

COST (LINK) /a/ {LINK} :R+ Every LINK has a
transportation cost rate.

FLOW (LINK) /va/ {LINK} :R+ There can be a non-
negative transportation FLOW (in tons) over each
LINK.

TOTAL (COST,FLOW) /f/ ;@SUMiSUMj(COSTij*FLOWij)
There is a TOTAL COST associated with all the FLOWS.

T:SUP (FLOW,SUPPLY) /t/ {PLANT} ;@SUMj(FLOWij) <=
SUPi Is the total FLOW leaving a PLANT less than
or equal to the SUPPLY CAPACITY.

T:DEM (DEMAND,FLOW) /t/ {CUST} ;@SUMi(FLOWij) =
DEMj  Is the total FLOW arriving at the customer
exactly equal to the its DEMAND.

NON_NEG (FLOW) /t/ ;FLOWij >= 0  This is the non-
negativity constraint.  The FLOW must be greater than
or equal to zero.
```

Figure 15: SML-based schema for the transportation problem after editing

## F. SUMMARY

We have presented the implementation details for LP/SM and described the conversion of the transportation problem. Because YACC will allow input from a very general class of grammars, we could extend these procedures to convert an LP model described in any mathematical language that can be represented in BNF.

The prototype described here, while sufficient to demonstrate the conversion propositions described by Dolk [Ref. 12], is not robust enough to serve as part of an MMS. The next section will discuss the limitations of our prototype and possibilities for future enhancements.

## VI. CONCLUSIONS

Our prototype clearly demonstrates it is possible to construct a structured modeling representation of simple LP models with a minimum amount of input from the user, however, this version is limited and a number of enhancements are required. Here we discuss the limitations of the current version of the prototype and the enhancements that would be required to produce a working MMS for LP models.

### A. LIMITATIONS OF THE PROTOTYPE

The primary limitation of our prototype parser is the parser's inability to "understand" the semantic rules of the mathematical language. The parser must be extended to include more than syntax checking. These semantic rules are necessary to ensure the validity of the model description in the model base.

The graphics presentation of the genus graph in our prototype is somewhat limited. In part, this is because of the difficulty in providing a general algorithm that will arrange the nodes and arcs in a manner that is consistently aesthetically pleasing. We suggest development of a system that allows the user to manipulate the nodes and arcs of the graph after it is drawn to improve the presentation of the graph. This would require saving genus position information

in the model base but would improve the capability of the system to communicate the structure of the model.

The quality of the graphics output could be improved through the use of the full VGA capabilities of the IBM PS/2. This would increase the screen resolution to 640x480, thus eliminating the jagged appearance of diagonal arcs. Bit-mapped scaleable characters would improve the quality of the textual information on the graph by allowing more precise placement. The presentation of the graph may be improved by assigning a distinct color and icon to each genus type as suggested by Wyant [Ref. 16].

The present display of the SML-based schema is limited to a single screen. A Scrollable text window would improve the presentation of the schema and allow for the display of larger models.

Although this prototype was designed with a multiple model, model base in mind, the present version of the prototype supports only a single model, model base. This could easily be extended to support multiple models as described by Dolk [Ref. 6:p. 714].

## B. AREAS FOR FURTHER RESEARCH

Representing structured models as data in a RDBMS is a promising approach for implementing the new generation of modeling systems described by Geoffrion. However, a number of research questions remain before a viable MMS can be constructed:

1. How can the model's description be integrated with the corresponding elemental detail ?

2. How can the model and its associated elemental detail be integrated with LP solvers ?

3. How can facilities be developed to allow ad hoc queries of the model base ?

Much remains to be done, but the potential benefits are worthy of the effort. A structured modeling based integrated modeling environment would provide decision makers with a better understanding of the models upon which their decisions are based. This would improve the acceptance of model-based systems by managers and enhance the quality of organizational decision making.

## APPENDIX A

## DESCRIPTION OF THE MATHEMATICAL LANGUAGE

### A. INTRODUCTION

This grammar is a subset of the Structured Modeling Language (SML) grammar proposed by Geoffrion [Ref. 10]. This language uses the Module Test Expression grammar and the Function Test Expression grammar to implement simple linear programming models. The following are the major features of SML Generic Rule grammar not supported by this language subset:

1. User defined functions
2. Standard functions FLOOR, MAX or MIN
3. Symbolic parameters
4. ORD function
5. Logical functions: @AND,@OR,@NOT
6. @IF function

### B. DESCRIPTION OF THE LANGUAGE SYNTAX

The grammar for our language subset is presented in Extended BNF form. The following conventions are used:

1. non-terminal symbols are enclosed in < >
2. optional items are enclosed in [ ]

3. items occurring zero or more times are enclosed in { }

4. alternative rules are separated by |

5. terminal symbols are enclosed in " "

## C. ALPHABET OF THE LANGUAGE

```
<divide_sign>       ::=  "/"
<exp_sign>          ::=  "^"
<minus_sign>        ::=  "-"
<plus_sign>         ::=  "+"
<mult_sign>         ::=  "*"
<at>                ::=  "@"
<colon>             ::=  ":"
<digit>             ::=  "0" | "1" | ....| "9"
<dollar>            ::=  "$"
<eq>                ::=  "="
<gt>                ::=  ">"
<ge>                ::= ">="
<lbracket>          ::=  "["
<literal>           ::= "#TRUE" | "#FALSE"
<lletter>           ::=  "a" | "b" | .... | "z"
<uletter>           ::=  "A" | "B" | .... | "Z"
<lparen>            ::= "("
<lt>                ::= "<"
```

```
<le>                    ::= "<="

<ne>                    ::= "<>"

<period>                ::=  "."

<p_digit>               ::= "1" | "2" | .... | "9"

<rbracket>              ::= "]"

<rparen>                ::= ")"

<rprime>                ::= "'"

<underscore>            ::= "_"

<zero>                  ::= "0"
```

## D. CHARACTERS USED IN IDENTIFIERS

```
<identifier_char>       ::= <digit>
                        | <uletter>
                        | <underscore>
```

## E. ARITHMETIC CONSTANTS

```
<sign>                  ::= <plus_sign>
                        | <minus_sign>

<p_integer>             ::=  <p_digit> {digit}

<nn_integer>            ::=  <p_digit>{digit} | <zero>

<nz_integer>            ::=  <p_integer> | <sign><p_integer>

<nn_real>               ::= <digit>{digit}<period>
                            <digit>{digit}
```

## F. NAMES OF MODEL ELEMENTS

```
<end>                   ::= "END"

<gname>                 ::= <uletter>{identifier_char}

                        | <dollar>{identifier_char}

<index>                 ::= <lletter>
```

```
<fc_name>               ::=   "ABS"|"EXP"|"LN"|"LOG"|"SQRT"

<i_fc_name>             ::= "SUM"
```

**G. CONTEXT FREE GRAMMAR FOR THE LANGUAGE**

```
<sp_index>              ::= <index><rprime>

<dp_index>              ::= <index><rprime><rprime>

<pp_index>              ::= <index>
                          | <sp_index>
                          | <dp_index>

<index_range4>          ::= <lbracket><pp_index><plus_sign>
                              <p_integer><colon>
                              <nz_integer><rbracket>

                          | <lbracket><pp_index><plus_sign>
                            <p_integer><colon><pp_index>
                            <plus_sign><p_integer><rbracket>

<index_range3>          ::= <lbracket><pp_index><minus_sign>
                              <p_integer><colon>
                              <nz_integer><rbracket>

                          | <lbracket><pp_index><minus_sign>
                            <p_integer><colon>
                            <pp_index><rbracket>

                          | <lbracket><pp_index><minus_sign>
                            <p_integer><colon><pp_index>
                            <sign><p_integer><rbracket>

<index_range2>          ::= <lbracket><pp_index><colon>
                              <nz_integer><rbracket>

                          | <lbracket><pp_index><colon>
                            <pp_index><plus_sign><p_integer>
                            <rbracket>

<index_range1>          ::= <lbracket><nz_integer><colon>
                            <nz_integer><rbracket>
```

```
                        |<lbracket><nz_integer><colon>
                         <pp_index><rbracket>

                        |<lbracket><nz_integer>
                         <colon><pp_index>
                         <sign><p_integer><rbracket>

<index_range>           ::=  EMPTY

                        | <index_range1>

                        | <index_range2>

                        | <index_range3>

                        | <index_range4>

<index_unit>            ::= <pp_index><index_range>

<iterated_fun_unit>     ::= <i_fc_name><index_unit>

                        | <iterated_fun_unit><i_fc_name>
                          <index_unit>

<index_sup_function>    ::= <at> <iterated_fun_unit><lparen>
                            <expression><rparen>

<offset_index>          ::= <lbracket><pp_index><sign>
                            <p_integer><rbracket>

<replaced_index>        ::= <lbracket><p_integer><rbracket>

                        | <lbracket><sign><p_integer>
                          <rbracket>

<gr_index>              ::= <pp_index>

                        | <replaced_index>

                        | <offset_index>

<gr_indicies>           ::= <gr_index>

                        | <gr_indicies><gr_index>
```

```
<simple_var>             ::=  <gname>
                         | <gname><gr_indicies>

<exprpack>               ::= <lparen><expression><rparen>

<built_in_function>      ::= <at><fc_name><exprpack>

<variable>               ::= <simple_var>
                         | <built_in_function>
                         | <index_sup_function>

<constant>               ::= <nn_integer>
                         | <nn_real>

<factor>                 ::=  <constant>
                         | <variable>
                         | <lparen><expression><rparen>

<power>                  ::=  <factor>
                         | <factor><exp_sign><power>

<term>                   ::=  <power>
                         | <term><mult_sign><power>
                         | <term><divide_sign><power>

expression               ::=  <term>
                         | <minus_sign><term>
                         | <expression><sign><term>

<function_expression>    ::= <expression>
```

```
<relational_operator>   ::=  <lt>
                           | <le>
                           | <eq>
                           | <gt>
                           | <ge>
                           | <ne>

<test_expression>     ::= <literal>
                        | <expression><relational_operator>
                          <expression>
                        | <expression><relational_operator>
                          <expression><relational_operator>
                          <expression>

<mod_test_expression>     ::=  <test_expression>

<f_t_expression>          ::=  <test_expression>
                             | <function_expression>
                             | <end>

<mod_function_expression>   ::=  <f_t_expression>
```

# APPENDIX B

# INPUT TO LEX AND YACC

## A. INPUT TO LEX

```
%{
/*

  These regular expression define all the symbols
  that are allowed in the mathematical language
*/

/* include all the manifest constants
        created by YACC for tokens */

#include "ytab.h"

/* include all symbol table data definitions */

#include "symbol.h"


/* define return value */
#define token(x) x


%}

%%

[\n]                 return 0;
[ \t]                     ;

"END"                {
                     install(yytext,END,eqno);
                     return token(END);
                     }

"SUM"                {
                     install(yytext,SUM,eqno);
                     return token(SUM);
                     }
```

```
#TRUE"                 {
                       install(yytext,LITERAL,eqno);
                       return token(LITERAL);
                       }

"#FALSE"               {
                       install(yytext,LITERAL,eqno);
                       return token(LITERAL);
                       }

"ABS"                  {
                       install(yytext,ABS,eqno);
                       return token(ABS);
                       }

"EXP"                  {
                       install(yytext,EXP,eqno);
                       return token(EXP);
                       }

"LOG"                  {
                       install(yytext,LOG,eqno);
                       return token(LOG);
                       }

"SQRT"                 {
                       install(yytext,SQRT,eqno);
                       return token(SQRT);
                       }

"LN"                   {
                       install(yytext,LN,eqno);
                       return token(LN);
                       }


[$A-Z][A-Z_]*                  {
                       install(yytext,IDENTIFIER,eqno);
                       return token(IDENTIFIER);
                       }

[a-z]*                 {
                       install(yytext,INDEX,eqno);
                       return token(INDEX);
                       }

[1-9][0-9]*            {
                       install(yytext,NZ_INTEGER,eqno);
                       return token(NZ_INTEGER);
                       }
```

```
[0-9][0-9]*          {
                     install(yytext,P_INTEGER,eqno);
                     return token(P_INTEGER);
                     }

[0-9]*"."[0-9]*          {
                     install(yytext,REAL,eqno);
                     return token(REAL);
                     }


":"                  {
                     install(yytext,COLON,eqno);
                     return token(COLON);
                     }

"@"                  {
                     install(yytext,AT,eqno);
                     return token(AT);
                     }

"*"                  {
                     install(yytext,TIMES,eqno);
                     return token(TIMES);
                     }

"/"                  {
                     install(yytext,DIVIDE,eqno);
                     return token(DIVIDE);
                     }

"-"                  {
                     install(yytext,MINUS,eqno);
                     return token(MINUS);
                     }

"+"                  {
                     install(yytext,PLUS,eqno);
                     return token(PLUS);
                     }

"^"                  {
                     install(yytext,POW,eqno);
                     return token(POW);
                     }

"("                  {
                     install(yytext,LPAREN,eqno);
                     return token(LPAREN);
                     }
```

```
")"                  {
                     install(yytext,RPAREN,eqno);
                     return token(RPAREN);
                     }

"["                  {
                     install(yytext,LBRACKET,eqno);
                     return token(LBRACKET);
                     }

"]"                  {
                     install(yytext,RBRACKET,eqno);
                     return token(RBRACKET);
                     }

"'"                  {
                     install(yytext,RPRIME,eqno);
                     return token(RPRIME);
                     }

"="                  {
                     install(yytext,EQ,eqno);
                     return token(EQ);
                     }

"<>"                 {
                     install(yytext,NE,eqno);
                     return token(NE);
                     }

"<"                  {
                     install(yytext,LT,eqno);
                     return token(LT);
                     }

">"                  {
                     install(yytext,GT,eqno);
                     return token(GT);
                     }

"<="                 {
                     install(yytext,LE,eqno);
                     return token(LE);
                     }

">="                 {
                     install(yytext,GE,eqno);
                     return token(GE);
                     }
```

```
%%
/* Symbol table routines
   functions that maintain the global symbol table used
   by parser.  The symbol table is defined as a singlely
   linked list
*/


extern Symbol *head,*tail;

/**********************/
Symbol *  install(s,t,e)

char *  s;   /* name of symbol */
int     t;   /* Yacc code for symbol type */
int     e;   /* equation symbol was found in */

{
   Symbol *  sp;       /* temp pointer to new symbol */



   /* dynamically allocate space for next symbol table
      entry and the space for the symbol name */

   sp = (Symbol *) malloc(sizeof(Symbol));

   sp->s_name = malloc(strlen(s)+1);


   /* assign values to the symbol table entry */

   strcpy(sp->s_name,s);
   sp->s_type = t;
   sp->equation = e;
   sp->next = NULL;


   /* add new symbol to end of linked list */


   if ( head == NULL ) {         /* first symbol on list */
        head = tail = sp;
        return;
   }

   else {                     /* add to end of list */
             tail->next = sp;
        tail = sp;
        return;
   }
}
```

## B. INPUT TO YACC

```
/* Mathematical Language syntax analysis */



/* Terminal Symbols */

%token  ABS
%token  AT
%token  COLON
%token  DIVIDE
%token  EQ
%token  EXP
%token   END
%token  GT
%token  GE
%token  IDENTIFIER
%token  INDEX
%token   LBRACKET
%token  LITERAL
%token   LT
%token  LE
%token  LPAREN
%token  LN
%token  LOG
%token  MINUS
%token  NE
%token  NZ_INTEGER
%token  POW
%token  P_INTEGER
%token  PLUS
%token  REAL
%token  RPAREN
%token  RBRACKET
%token   RPRIME
%token  SUM
%token  SQRT
%token  TIMES


/* define the precedence of the operators */

%left   PLUS MINUS
%left   TIMES DIVIDE
%left   UNARYMINUS
%right  POW
```

```
%%

equation            : test_expression
                    | function_expression
                    | END

test_expression     : LITERAL
                    | expression rel_op expression
                    | expression rel_op expression rel_op
expression

rel_op              : LE
                    | GT
                    | GE
                    | EQ
                    | NE
                    | LT

function_expression    : expression

expression             : term
                       | expression PLUS term
                       | expression MINUS term
                       | MINUS term    %prec UNARYMINUS

term                   : power
                       | term TIMES power
                       | term DIVIDE power

power                  : factor
                       | factor POW power

factor                 : constant
                       | variable
                       | exprpack

constant               : integer
                       | REAL

variable               : simple_variable
                       | built_in_function
                       | index_sup_function

built_in_function      : AT builtin exprpack

builtin                : ABS
                       | EXP
```

```
                        | SQRT
                        | LOG
                        | LN

exprpack                : LPAREN expression RPAREN

simple_variable         : IDENTIFIER
                        | IDENTIFIER gr_indicies

gr_indicies             : gr_index
                        | gr_indicies gr_index

gr_index                : pp_index
                        | replaced_index
                        | offset_index

pp_index                : INDEX
                        | INDEX RPRIME
                        | INDEX RPRIME RPRIME

replaced_index     : LBRACKET P_INTEGER RBRACKET
                   | LBRACKET PLUS P_INTEGER RBRACKET
                   | LBRACKET MINUS P_INTEGER RBRACKET

offset_index       : LBRACKET pp_index PLUS P_INTEGER
                         RBRACKET
                   | LBRACKET pp_index MINUS P_INTEGER
                         RBRACKET

index_sup_function      : AT iterated_fun_unit exprpack

iterated_fun_unit       : SUM  index_unit
                        | iterated_fun_unit SUM index_unit

index_unit              : pp_index index_range

index_range             :         /* no range */
                        | index_range1
                        | index_range2
                        | index_range3
                        | index_range4

index_range1       : LBRACKET NZ_INTEGER COLON NZ_INTEGER
                         RBRACKET

                   | LBRACKET NZ_INTEGER COLON pp_index
                         RBRACKET
                   | LBRACKET NZ_INTEGER COLON pp_index sign
                         integer  RBRACKET

index_range2       : LBRACKET pp_index COLON NZ_INTEGER
                        RBRACKET
```

```
                    | LBRACKET pp_index COLON pp_index PLUS
                        integer RBRACKET

index_range3        : LBRACKET pp_index MINUS P_INTEGER COLON
                        NZ_INTEGER RBRACKET
                    | LBRACKET pp_index MINUS P_INTEGER COLON
                        pp_index RBRACKET
                    | LBRACKET pp_index MINUS P_INTEGER COLON
                        pp_index sign integer RBRACKET

index_range4        : LBRACKET pp_index PLUS integer COLON
                        NZ_INTEGER RBRACKET
                    | LBRACKET pp_index PLUS integer COLON
                        pp_index PLUS integer RBRACKET

integer             : P_INTEGER
                    | NZ_INTEGER

sign                : PLUS
                    | MINUS
%%
```

## APPENDIX C

## MAKEFILE AND SOURCE LISTING

```
##############################################
#
#  Makefile for XMMS prototype
#
#   A prototype parser that converts a LP
#   mathematical language into structured modeling
#   formats.  The models are stored in a ORACLE
#   database.  Models can be displayed as textual schema
#   or as generic graph.


#   Writen By: David S. Hill

#   Uses large memory model because of
#   ORACLE PRO*C interface

MODEL=L

# Object Files

OBJS =parser.obj menu.obj models.obj enter.obj oracle_r.obj
oracle_w.obj printem.objA

# Complier Flags

CFLAGS =/A$(MODEL) /c

CL =cl $(CFLAGS)

# General Rule -
# make .obj from .c files

.C.OBJ:
   $(CL) $*.c

# Compile all the files

# Parser program created by input to YACC  and LEX

parser.obj:  parser.c ytab.h

# Menu function and main program
```

```
menu.obj:  menu.c

# functions display models

models.obj:  models.c
# function to enter model

enter.obj:  enter.c

# function to read and write from ORACLE

oracle_r.c:  oracle_r.pc
  pcc iname=oracle_r.pc host=c

oracle_r.obj:  oracle_r.c

oracle_w.c:  oracle_w.pc
  pcc iname=oracle_w.pc host=c

oracle_w.obj:  oracle_w.c

/* link the executable fle */

xmms.exe:  $(OBJS)
  LINK $(OBJS),xmms.exe,,sqlmsc /se:512 /stack:10000 /map;
```

```
/*****************************************************
DATE:          10 Jan 89 - dsh

FILE:          defs.h

CONTENTS:      Constant definitions for Keys, Colors and other
               common constants
*****************************************************/

/* Constants */

#define DEFAULT    -1
#define TRUE       1
#define FALSE      0


/* Key Scan codes */

#define UP         72
#define DOWN       80
#define LEFT       75
#define RIGHT      77
#define ENTER      28

/* text color definations */

#define  BLACK     0
#define  BLUE      1
#define  CYAN      3
#define  RED       4
#define  WHITE     7
#define  LBLUE     9
#define  LRED      12

#define  BRWHITE   15

/* video interrupt defs */

#define CURSIZE    1         /* set cursor size service
number */
#define  VIDEO     0x10      /* interrupt number */
#define  OFFBIT    0x20      /* this bit turns cursor off
                             sets bit 5 of register ch on   */
```

```
/***********************************************************
DATE:        5 Dec 88 - dsh

FILE:        ytab.h

CONTENTS:    Constant definitions passed from scanner
              to parser.  Used in manipulating symbol table
***********************************************************/

# define ABS 257
# define AT 258
# define COLON 259
# define DIVIDE 260
# define EQ 261
# define EXP 262
# define END 263
# define GT 264
# define GE 265
# define IDENTIFIER 266
# define INDEX 267
# define LBRACKET 268
# define LITERAL 269
# define LT 270
# define LE 271
# define LPAREN 272
# define LN 273
# define LOG 274
# define MINUS 275
# define NE 276
# define NZ_INTEGER 277
# define POW 278
# define P_INTEGER 279
# define PLUS 280
# define REAL 281
# define RPAREN 282
# define RBRACKET 283
# define RPRIME 284
# define SUM 285
# define SQRT 286
# define TIMES 287
# define UNARYMINUS 288
```

```
/*************************************************************
DATE:         10 Jan 89 - dsh

FILE:         symbol.h

CONTENTS:     Type definitions and declarations for all
              global linked list elements
************************************************************/

typedef struct symbol_table_entry       Symbol;


struct symbol_table_entry        {

   char       s_name[20];     /* name of symbol */
   int        s_type;         /* YACC code for symbol type */
   int        equation;       /* equation # symbol was in */
   struct symbol_table_entry  * next;
                          /* pointer to next symbol in list */

};


typedef struct entity_table_entry     Entity;


struct entity_table_entry   {

   char ename[20];            /* name of entity  */
   char etype[8];             /* type of entity */
   char dname[30];            /* descriptive name of entity */
   char idx[4];               /* index set */
   char idx_stmt[50];         /* index statement */
   char grange[20];           /* generic range stmt */
   char grule[80];            /* generic rule */
   char comments[80];         /* informal interpertation */
   struct entity_table_entry *next;   /* pointer to next
                                 table entry */
};

typedef struct relship_table_entry   Relship;

struct relship_table_entry  {
   char       rtype[12];         /* type of relationship
                                     CALLS  */
   char       e1name[20];        /* calling entity name */
   char       e1type[8];         /* calling entity type */
   char       e2name[20];        /* called entity name */
   char       e2type[8];         /* called entity type */
struct relship_table_entry *next;     /* pointer to next
                                 table entry */
};
```

```
typedef struct module_table_entry    Module;

struct module_table_entry   {
  char rtype[12];                /* type of relationship
                                    CONTAINS */

  char      e1name[20];          /* calling entity name */
  char      e1type[8];           /* calling entity type */
  char      e2name[20];          /* called entity name */
  char      e2type[8];           /* called entity type */
  int       rel_pos;             /* Position in heirarchy */
  struct module_table_entry *next;   /* pointer to next
                                 table entry */

};

typedef struct entity_position Position;


struct entity_position {

char    ename[20];          /* name of entity */
char    etype[8];           /* type of entity */
short   xpos,               /* x pixel coordinate */
        ypos,               /* y pixel coordinate */
        level;              /* genus graph level - all
                            pe's start
                            on level 1 and those
                            elements that call
                            them are on the next level */
struct entity_position * next;  /* pointer to next item
                                 in linked list */
};
```

```
/*************************************************************

DATE:          22 Jan 89 - dsh

FILE:          menu.c

CONTENTS:      Main function and the functions to control
               the menu selection.  The menu selection
functions were adapted from the basic graphics
samples provided with the Microsoft C compiler.
**********************************************************/

#include <stdio.h>      /* standard io library defs */
#include <dos.h>        /* defines registers
                        for interrupts */
#include "defs.h"       /* constant definations */
#include "symbol.h"     /* type definitions */
#include <graph.h>      /* graphics defs */
#include <string.h>     /* string function defs */
#include <bios.h>       /* defs for BIOS calls to keyboard */


/* Functions - prototypes */

int menu(int, int, char* []);
void box(int, int, int,    int);
void itemize(int, int, char*, int);
short initialize(short);
int yylex(int);
void yyerror(void);
extern void enter_model(void);
int yyparse(int);
void cursor_on(void);
void cursor_off(void);
void display_message(char *);


/* Structure for configuration    */

struct videoconfig vc;

/* Array and enum for main menu */

char *mnuMain[]  = {
   "Enter Model",
   "Edit Model Schema",
   "Display Model Graph",
   "Display Model Schema",
   "Quit",
   NULL };
```

```
enum {
   NEW, EDIT, GRAPH, SCHEMA, QUIT };


/* Structure for menu attributes
        (variables for color and monochrome) */

struct mnuAtr {
   int  fgNormal, fgSelect, fgBorder;
   long bgNormal, bgSelect, bgBorder;
   int  centered;
   char nw[2], ne[2], se[2], sw[2], ns[2], ew[2];
}
menus   = {
   BLACK, BRWHITE, RED,
   CYAN, RED, CYAN,
   TRUE,
   "\xda", "\xbf", "\xd9", "\xc0", "\xb3", "\xc4"
};

struct mnuAtr bwmenus =     {
   0x70, 0xf, 0x70,
   0x7, 0x70, 0x7,
   TRUE,
   "\xda", "\xbf", "\xd9", "\xc0", "\xb3", "\xc4"
};

char mess1[] = {
   "Prototype Model Management System for Linear
Programming" }
;

char mess2[] = {
   "Move to menu selection with cursor keys, press ENTER to
 elect" };

char mess3[] = {
   "A Model already exists do you want to delete it ?
(Y/N)"};

int     lmess1 = sizeof(mess1),
lmess2 = sizeof(mess2);
lmess3 = sizeof(mess3);
```

```
/************************************************************

                          MAIN

   Main function for XMMS.  Makes calls to set up menus,
and calls for each of the functions in the menu list.


************************************************************/


main ()
{
   int  choice,    /* menu choice */
   crow,           /* current text position row */
   ccol;           /* current text position col */

   int  tmode,     /* video text mode */
        vmode;     /* video mode for graphics */

   long bkcolor;   /* initial background color */



   short     xpixels,  /* max pixels in
                       x in graphics mode */
             ypixels,  /* max pixels in y
                       in graphics mode */
             maxcols,  /* max text columns in text mode */
             maxrows,  /* max text rows in text mode */
             mflag = 0;/* global flag set when
                       model is present */

   char      yorn[4];  /* buffer for y or n answer */

   _getvideoconfig(&vc);

   xpixels = vc.numxpixels;
   ypixels= vc.numypixels;
   maxcols= vc.numtextcols;
   maxrows= vc.numtextrows;

   crow = maxrows / 2;
   ccol = maxcols / 2;
```

```
/* Select best text and  graphics
          modes and adjust menus    */

switch (vc.adapter) {
case _MDPA :
case _CGA :
     puts("EGA or VGA Graphics required.\n");
     exit(0);
case _EGA :
case _VGA :
case _MCGA :
     vmode =   _ERESCOLOR;
     break;
}
switch (vc.mode) {
case _TEXTBW80 :
     menus =   bwmenus;
case _TEXTBW40 :
     _setvideomode(_TEXTBW80);
     break;
case _TEXTC40 :
     tmode =   _TEXTC80;
     break;
case _TEXTMONO :
case _ERESNOCOLOR :
     menus =   bwmenus;
     tmode =   _TEXTMONO;
     vmode =   _ERESNOCOLOR;
     break;
default   :
     tmode =   _TEXTC80;
}

_setvideomode(tmode);

/* delete old models if any exist in ORACLE */

delete_from_entity();
delete_from_relship();

_settextposition(2,40 -  (lmess1   / 2));
_outtext(mess1);
_settextposition(22,40 - (lmess2   / 2));
_outtext(mess2);
```

```
/* Select and branch to  menu choices */

for (;;) {
     choice = menu(crow,ccol,mnuMain);


     switch (choice) {
     case NEW :
          if(mflag) {
               _clearscreen(_GCLEARSCREEN);
               display_message("Single model prototype -
                         model exists ");
               break;
          }
          _clearscreen(_GCLEARSCREEN);
          enter_model(  );
          mflag = 1;
          break;
     case EDIT :
          if(mflag) {
               _clearscreen(_GCLEARSCREEN);
               edit_schema(  );
          }
          else {
               _clearscreen(_GCLEARSCREEN);
               display_message("Single model prototype -
                         enter model first ");
          }
          break;
     case GRAPH :
          if(mflag) {
               initialize(vmode);
               display_graph(  );
               _bios_keybrd(_KEYBRD_READ);
          }
          else {
               _clearscreen(_GCLEARSCREEN);
               display_message("Single model prototype -
                    enter model first ");
          }
          break;
     case SCHEMA :
          if(mflag) {
               _clearscreen(_GCLEARSCREEN);
               display_schema( );
               _bios_keybrd(_KEYBRD_READ);
          }
          else {
               _clearscreen(_GCLEARSCREEN);
               display_message("Single model prototype -
                    enter model first ");
          }
```

```
                break;
            case QUIT :
                _setvideomode (_DEFAULTMODE);
                exit(0);

            }
            _setvideomode (tmode);
        }
}


/*****************************************************
            MENU

    Function to put menu on screen.
    Returns number of item selected.
    Adapted from Microsoft C graphics samples.
*****************************************************/

int menu(row, col, items)
int     row,        /* starting row and col */
        col;
char    *items[];   /* array of menu items */
{
    int  i,
         num,
         max= 2,
         prev,
         curr= 0,
         choice;
    int  litem[25];
    long bcolor;

    cursor_off();

    bcolor = _getbkcolor();

    /* Count items,find longest,
         and put length of each in array */
    for (num = 0; items[num]; num++) {
         litem[num] = strlen(items[num]);
         max = (litem[num] > max) ? litem[num] : max;
    }
    max += 2;

    if (menus.centered) {
         row -= num / 2;
         col -= max / 2;
    }

    /* Draw   menu box */
```

```
    _settextcolor(menus.fgBorder);
    _setbkcolor(menus.bgBorder);
    box(row++,col++,num,max);

    /* Put items in     menu */

    for (i = 0; i <     num; ++i) {
         if (i == curr) {
              _settextcolor(menus.fgSelect);
              _setbkcolor(menus.bgSelect);
         }
         else {
              _settextcolor(menus.fgNormal);
              _setbkcolor(menus.bgNormal);
         }
         itemize(row+i,col,items[i],max - litem[i]);
    }

    /* Get selection */

    for (;;) {
         switch ((_bios_keybrd(_KEYBRD_READ) & 0xff00) >> 8)
         {
         case UP   :
              prev = curr;
              curr = (curr > 0) ? (--curr % num) : num-1;
              break;
         case DOWN :
              prev = curr;
              curr = (curr < num) ? (++curr %    num) : 0;
              break;
         case ENTER :
              _setbkcolor(bcolor);
              return(curr);
         default   :
              continue;
         }
         _settextcolor(menus.fgSelect);
         _setbkcolor(menus.bgSelect);
         itemize(row+curr,col,items[curr],max - litem[curr]);

         _settextcolor(menus.fgNormal);
         _setbkcolor(menus.bgNormal);
         itemize(row+prev,col,items[prev],max - litem[prev]);

    }
}
```

```
/*********************************************************

            BOX

   Draw menu box.
  <row> and <col> are upper left of box.
  <hi> and <wid> are height and width.
**********************************************************
/

void box(row, col, hi, wid)
int row, col, hi, wid;
{
  int i;
  char temp[80];

  _settextposition(row,col);
  temp[0] = *menus.nw;
  memset(temp+1,*menus.ew,wid);
  temp[wid+1] = *menus.ne;
  temp[wid+2] = NULL;
  _outtext(temp);
  for (i = 1; i <= hi; ++i) {
       _settextposition(row+i,col);
       _outtext(menus.ns);
       _settextposition(row+i,col+wid+1);
       _outtext(menus.ns);
  }
  _settextposition(row+hi+1,col);
  temp[0] = *menus.sw;
  memset(temp+1,*menus.ew,wid);
  temp[wid+1] = *menus.se;
  temp[wid+2] = NULL;
  _outtext(temp);
}

/********************************************************
            ITEMIZE

   Put an item in menu.
  <row> and <col> are left position.
  <str> is the string item.
  <len> is the number of blanks to fill.
********************************************************/

void itemize(row,col,str,len)
int row, col, len;
char str[];
{
  char temp[80];

  _settextposition(row,col);
```

```
    _outtext(" ");
    _outtext(str);
    memset(temp,' ',len--);
    temp[len] = NULL;
    _outtext(temp);
}

/*****************************************************

                          INITIALIZE

        Set the display mode <mode> is the mode to set.
    Returns 0 if  mode is  invalid, else returns nonzero.
*****************************************************/

short initialize(mode)
short mode;
{
    int  ret,
         i = 0,
         btm,
         top = 63,
         inc,
         red = 0,
         green = 0,
         blue = 0;

    _getvideoconfig(&vc);
    if (mode < _MRES4COLOR)
         return(0);
    if (!(mode == vc.mode))  {
         if(!(ret = _setvideomode(mode)))
              return(0);
         _getvideoconfig(&vc);
    }
    else
         ret = mode;

    _setlogorg(0,0);

    _moveto(0,0);

    return(ret);
}
```

```
/*****************************************************
                 CURSOR_ON

  This function uses the dos call to a POM
BIOS routine to set the size of the cursor

*****************************************************/
void cursor_on()

{

  union REGS     regs;      /* register values for
                            interrupts */

  int       start = 7,      /* starting scan line
                            of cursor */
            end = 8;        /* end scan line of cursor */

  /* set interrupt values */

  regs.h.ch = (char) start;
  regs.h.cl = (char) end;
  regs.h.ah = CURSIZE;

  /* interrupt call */
  int86(VIDEO,&regs,&regs);

  return;
}

/*****************************************************
            CURSOR_OFF

  This function uses the dos call to a ROM
BIOS routine to set the size of the cursor
*****************************************************/
void cursor_off()

{

  union REGS     regs;      /* register values for
                            interrupts */
  /* set register values */

  regs.h.ch = OFFBIT;
  regs.h.ah = CURSIZE;

  /* interrupt call */
  int86(VIDEO,&regs,&regs);
  return;

}
```

```
/**********************************************************

DATE:        16 Feb 89 - dsh

FILE:        models.c

CONTENTS:    Contains the functions for display of
       the schema and the genus graph.
*********************************************************/
#include "defs.h"       /* constant definations */
#include <malloc.h>     /* memory allocation function defs */

#include <stdlib.h>     /* std ansi lib defs */
#include <stdio.h>      /* standard i/o function defs */
#include <graph.h>      /* graphics defs     */
#include "symbol.h"     /* symbol table defs */
#include <bios.h>       /* defs for BIOS call to keyboard */
#include <string.h>     /* defs for strlen function */

#include <ctype.h>      /* defs for character type
                            functions */

#define MAX_LEVELS      8

/* function prototypes for the functi ns in this file */

void display_graph(void);
void display_schema(void);
void edit_schema(void);
void install_position(char*,char*,short);
void compute_positions(void);
void read_schema(void);
void place_text(void);
void draw_lines(void);
void draw_arrows(void);
void display_message(char*);
void build_calling_seq(char *,char *);
void compute_positions(void);


/* global head and tail for the Entity table linked list */

extern Entity * ehead;
extern Entity * etail;

/* global head and tail for the Relship table linked list */


extern Relship * rhead;
extern R lship * rtail;
```

```
/* head and tail for the Position table linked list */

Position * phead = NULL;
Position * ptail = NULL;

/* array to track the number of genera on each level */

short   number_on_level[MAX_LEVELS];

char * cont_msg = "Any key to return to Menu";

/**********************************************************

                    DISPLAY_GRAPH

  Function to display the genus graph.  Adapted from the
graphical interface work done by Wyant (Ref. 16)

**********************************************************/

void display_graph()

{
  struct videoconfig  config;    /* struct for return of
video config info */
  short            maxx,         /* max pixels on x axis */
                   maxy;         /* max pixels of y axis */
  Relship          *rp;          /* pointer to relship
                                  linked list */
  Position         *pp;          /* pointer to position
                                 linked list */
  int              level,        /* position level
                                 in graph */
                   length,       /* strlen of message */
                   begin,        /* beginning col
                                      of message */
                   i;        /* counter */

  _getvideoconfig(&config);

  maxx = config.numxpixels;
  maxy = config.numypixels;

  /* read in the call information from ORACLE table */

  read_relship();

  /* install all pe on level 1 */

  rp = rhead;
  level = 1;
```

```
while (rp != NULL)  {

     if (strcmp(rp->e2type,"pe") == 0)  {
          install_position(rp->e2name,rp->e2type,level);
     }

     rp = rp->next;
}

/* see if primitive entities exist
     in the model if not display error */

if (phead == NULL)  {
     display_message("Bad Model - no primitive
                    entities -> Please Renter");
     delete_from_entity();
     delete_from_relship();
     return;
}

rp = rhead;
pp = phead;

while (pp != NULL)  {

     /* check for all
     elements that call those on the position list */

     while(rp != NULL)   {

          if (strcmp(pp->ename,rp->e2name) == 0)  {

install_position(rp->e1name,rp->e1type,(pp->level)+1);
          }
          rp = rp->next;

     }

     pp = pp->next;
     rp = rhead;
}

compute_positions();

place_text();

draw_arrows();

draw_lines();
```

```
    /* display any key message */

    length = strlen(cont_msg);
    begin = 40 - (length/2);
    _settextcolor(BRWHITE);
    _settextposition(1,begin);
    _outtext(cont_msg);

    /* free memory */
    free_relship_list();
    free_position_list();
    free_entity_list();

    /* reset counters */

    for(i =0;i< MAX_LEVELS;i++)    {
         number_on_level[i] = 0;
    }

    return;

}

/*************************************************************
                   DISPLAY_MESSAGE

    opens a text window and displays the text passed in
    by the argument

*************************************************************/

void display_message(mess)

char    mess[50]; /* buffer to hold message to display */

{

    long       old_color;      /* old background color
                               to restore */
    short      old_text_color;/* color to restore */
    int        mlength;        /* length of message
                               for centering */
    char       buffer[80];     /* text buffer for continue
                               ... message */


    old_color = _getbkcolor();
    old_text_color = _gettextcolor();

    _settextwindow(15,5,10,70);
    _settextcolor(BRWHITE);
    _setbkcolor(CYAN);
```

```
    _clearscreen(_GWINDOW);

    /* center message */

    mlength = strlen(mess);
    _settextpcsition(2,30-(mlength/2));
    _outtext(mess);


    sprintf(buffer,"Any key to continue.........");

    mlength = strlen(buffer);
    _settextposition(4,30-(mlength/2));
    _outtext(buffer);

    _bios_keybrd(_KEYBRD_READ);

    /* restore the screen */



    _settextcolor(old_text_color);
    _setbkcolor(old_color);
    _clearscreen(_GWINDOW);

    _settextwindow(1,1,25,80);

    return;

}

/************************************************************
                       INSTALL_POSITION

    This function places each entity found in display_graph
into the position linked list. If the element is already on
the list on the same display level it will not be added.  If
it is already on the list on a lower display level the lower
level is converted to a place holder to allow the arc to
span levels

************************************************************/
void install_position(ename,etype,level)

char    ename[20];      /* name of element */
char    etype[8];       /* type of element */
short   level;          /* the display level
                        of the element */

{
    Position   *pp, /* temp pointer to new structure */
               *tp; /* temp pointer to walk through
```

```
                the linked list */

/* create new structure */

pp = (Position *) malloc (sizeof(Position));

strcpy(pp->ename,ename);
strcpy(pp->etype,etype);
pp->level = level;
pp->next = NULL;

/* if list is empty - add it */

if (phead == NULL)  {
     phead = ptail = pp;
     number_on_level[level]++;
     return;
}

/* check to see if it is there already */

tp = phead;

while(tp != NULL)   {

     if (strcmp(ename,tp->ename) == 0)  {

          /* see if it is on a lower level
             if so we will make the lower level
               a space holder and add new position
               on higher level to list */

          if(tp->level < level)    {
               number_on_level[level]++;
               strcpy(tp->etype,"sp");
               ptail->next = pp;
               ptail = pp;
               return;
          }
          /* don't add second position if it
          is already on the list on this level */

          else {

               free(pp);
               return;
          }

     }
     /* not same ename so look through
          the rest of list */
```

```
        tp = tp->next;

    }

    /* not on list so add to end */
    number_on_level[level]++;
    ptail ->next = pp;
    ptail = pp;
    return;


}

/************************************************************

            COMPUTE_POSITIONS

    Function to compute the x and y coordinates of the
    elements of the genus graph


************************************************************/

void compute_positions()

{

    Position  *pp; /* temp pointer into position list */

    short     maxx,     /* max pixels in x direction */
              maxy,     /* max pixels in y direction */
              max_level,
                        /* max number of levels */
              xpos,     /* x position of element */
              ypos,     /* y position of element */
              xint,     /* interval between elements
                        on level */
              yint,     /* interval between levels */
              level;    /* counter for current level */

    struct videoconfig  vc;
    /* return variable that holds video
                   config info */

    _getvideoconfig(&vc);

    maxx = vc.numxpixels;
    maxy = vc.numypixels;


    max_level = ptail->level;
```

```
   level = 1;

   /* initialize pointer to head of list */

   pp = phead;

   /* space genera evenly on screen */

   yint = (short) (maxy/(max_level + 1));

   xint = (short) (maxx/(number_on_level[level]+1));

   ypos = yint;

   xpos = xint;

   while(pp != NULL)   {

        if(pp->level > level)    {
             level++;
             xint = (short) (maxx
/(number_on_level[level]+1));
             ypos += yint;
             xpos = xint;
        }

        /* assign position */

        pp->xpos = xpos;

        pp->ypos = ypos;

        /* increment x position  and go to next position */

        xpos += xint;

        pp = pp->next;

   }

}
/************************************************************
                         PLACE_TEXT

   Function to place the genus name on the genus graph

************************************************************/
void place_text()

{
```

```
Position  *pp; /* temp pointer into position list */

struct videoconfig  vc;
/* return variable that holds video
               config info */

short
               length,   /* length of string to center */

               maxx,     /* max pixels in x direction */
               maxy;     /* max pixels in y direction */

     int       centerx,  /* col on which to
                         center text */
               centery;  /* row on which to
                         center text */

float          textx = 80.0,  /* max values
                              of text coordinates */
               texty = 28.0;  /* must be floats to
                              do coordinate
                              conversion */

char           type_buf[6];   /* buffer to hold
                              entity type */


_getvideoconfig(&vc);

maxx = vc.numxpixels;
maxy = vc.numypixels;

pp = phead;

while(pp != NULL)    {

     /* convert graphics position to text row and col */

     centerx = (short) (textx * (pp->xpos) /maxx);
     centery = (short) (28 - (texty * (pp->ypos)/maxy));

     length = strlen(pp->ename);

     _settextposition(centery,(centerx - (length /2)));

     if (strcmp(pp->etype,"sp") != 0)    {

          _outtext(pp->ename);
          sprintf(type_buf," /%s/",pp->etype);
          _outtext(type_buf);
     }
```

```
        pp = pp->next;

    }

}
/************************************************************

            DRAW_ARROWS

    Function to draw the arrow heads for lines representing
     the calls on the genus graph

************************************************************/

void draw_arrows()

{
    short           x,          /* x pos of arrow tip */
                    y,          /* y pos of arrow tip */
                    maxx,       /* max pixels in x direction */
                    maxy;       /* max pixels in y direction */

    Position   *pp; /* temp pointer to the position
                    linked list */

    struct videoconfig  vc;
    /* return variable that holds video
                    config info */

    _getvideoconfig(&vc);

    maxx = vc.numxpixels;
    maxy = vc.numypixels;

    pp = phead;

    while(pp != NULL)    {
         /* draw arrowheads on all positions
              except spaceholders and pe */

         if (strcmp(pp->etype,"sp") != 0)    {

              if(strcmp(pp->etype,"pe") != 0)     {
                   x = pp->xpos;
                   /* offset y  from the text */
                   y = pp->ypos - 15;

                   /* draw arrow head */
                   _moveto(x,maxy-y);
                   _lineto(x-3,maxy-y+4);
                   _lineto(x+3,maxy-y+4);
```

```
                    _lineto(x,maxy-y);
                    _floodfill(x,maxy-y+2,BRWHITE);
                    _moveto(x,maxy-y+4);
                    _lineto(x,maxy-y+8);

               }
          }
          pp = pp->next;
     }
     return;
}


/**********************************************************

            DRAW_LINES

     Function to draw the lines representing the calls on
     the genus graph

**********************************************************/
void draw_lines()

{

     Relship        *rp; /* temp pointer to the
                          relship linked list */
     Position       *pp; /* temp pointer to the
                          position linked list */
     Position       *tp; /* pointer to find
                          all calling elements */

     short          level_diff,    /* vertical difference
                                   between two genus levels */
                    begin,         /* begining level
                                   of line */
                    xbegin,        /* x coordinate of
                                        called element */
                    ybegin,        /* y coordinate of
                                    called element */
                    xend,          /* x coordinate of
                                   calling element */
                    yend,          /* y coordinate
                                   of  calling element */
                    maxx,          /* max pixels in x
direction */
                    maxy;          /* max pixels in y
direction */


     struct videoconfig  vc;
```

```
struct videoconfig  vc;
/* return variable that holds video
               config info */

char      called_name[20], /* temp buffers
                              for the strings to
                              compare */
          calling_name[20];

/* initailize pointers */

rp = rhead;

_setcolor(BRWHITE);

_getvideoconfig(&vc);

maxx = vc.numxpixels;
maxy = vc.numypixels;

while (rp != NULL)  {

     pp = phead;

     strcpy(called_name,rp->e2name);

     strcpy(calling_name,rp->e1name);


     while (pp != NULL)  {

          /* find position of called element */

          if(strcmp(called_name,pp->ename) == 0)  {

               begin = pp->level;
               xbegin = pp->xpos;
               ybegin = pp->ypos;

               /* lock through the rest of list
                          for all calling elements */

               tp = pp->next;

               while(tp != NULL)   {

               if(strcmp(calling_name,tp->ename) == 0) {

                         level_diff = tp->level - begin;

                         if(level_diff == 1) {
```

```
        /* leave space for text if not space holder */
                       if (strcmp(tp->etype,"sp") == 0)

                                 yend = tp->ypos;
                            else
                                 yend = tp->ypos - 25;

                            _moveto(pp->xpos,
                                      maxy - pp->ypos);

                            _lineto(xend,maxy-yend);

                       }

                       if (level_diff > 1) {
             /* draw upper part from space holder */
                            xbegin = xend;
                            ybegin = yend;
                            xend = tp->xpos;
                            yend = tp->ypos-25;
                            _moveto(xbegin,
                                      maxy-ybegin);
                            _lineto(xend,maxy-yend);
                       }
                  }

                  tp = tp->next;

             }

        }
        pp = pp->next;
    }
    rp = rp ->next;
  }
}
```

```
/*********************************************************
            DISPLAY_SCHEMA

  Function to display the model's structured modeling
  schema.

**********************************************************/

void display_schema()

{

  char buffer[240],         /* buffer to hold genus para */
  calling_seq[80];          /* buffer to construct
                            calling sequence */

  struct rccoord text;      /* current text row and col */

  int        length,        /* strlen of message */
             begin;         /* beginning col of message */

  Entity     * ep;          /* pointer to linked list */


  /* read in data */

  read_schema();

  read_relship();

  /* set up screen */

  _settextwindow(1,1,25,80);

  _wrapon(_GWRAPON);

  _settextcolor(BRWHITE);


  /* initialize pointer to top of list */

  ep = ehead;

  while(ep != NULL)    {

       if (strcmp(ep->etype,"model") == 0)     {

            text = _gettextposition();
            text.row += 2;
            text.col = 0;
            _settextposition(text.row,text.col);
```

```
        sprintf(buffer,"%s
          %s.",ep->ename,ep->comments);
        _outtext(buffer);
}

else if (strcmp(ep->etype,"pe") == 0)    {

        text = _gettextposition();
        text.row += 2;
        text.col = 0;
        _settextposition(text.row,text.col);
        sprintf(buffer,"%s /pe/  %s.",
        ep->ename,ep->comments);
        _outtext(buffer);
}

else if (strcmp(ep->etype,"ce") == 0)    {

        text = _gettextposition();
        text.row += 2;
        text.col = 0;
        _settextposition(text.row,text.col);

        build_calling_seq(ep->ename,calling_seq);

        sprintf(buffer,"%s %s /ce/ %s  %s.",
        ep->ename,calling_seq,ep->idx_stmt,
        ep->comments);
        _outtext(buffer);
}

else if (strcmp(ep->etype,"a") == 0)    {

        text = _gettextposition();
        text.row += 2;
        text.col = 0;
        _settextposition(text.row,text.col);

        build_calling_seq(ep->ename,calling_seq);

        sprintf(buffer,"%s %s /a/ %s %s  %s.",
        ep->ename,calling_seq,ep->idx_stmt,
        ep->grange,ep->comments);
        _outtext(buffer);
}

else if (strcmp(ep->etype,"va") == 0)    {

        text = _gettextposition();
        text.row += 2;
        text.col = 0;
        _settextposition(text.row,text.col);
```

```
            build_calling_seq(ep->ename,calling_seq);

            sprintf(buffer,"%s %s /va/ %s %s  %s.",
            ep->ename,calling_seq,ep->idx_stmt,
            ep->grange,ep->comments);
            _outtext(buffer);
        }


        else if (strcmp(ep->etype,"t") == 0)    {

            text = _gettextposition();
            text.row += 2;
            text.col = 0;
            _settextposition(text.row,text.col);

            build_calling_seq(ep->ename,calling_seq);

            sprintf(buffer,"%s %s /t/ %s ;%s  %s.",
            ep->ename,calling_seq,ep->idx_stmt,
            ep->grule,ep->comments);
            _outtext(buffer);
        }

        else if (strcmp(ep->etype,"f") == 0)    {

            text = _gettextposition();
            text.row += 2;
            text.col = 0;
            _settextposition(text.row,text.col);

            build_calling_seq(ep->ename,calling_seq);

            sprintf(buffer,"%s %s /f/ %s ;%s  %s.",
            ep->ename,calling_seq,ep->idx_stmt,
            ep->grule,ep->comments);
            _outtext(buffer);

        }
        ep = ep->next;

    }

    /* display any key message */

    length = strlen(cont_msg);
    begin = 40 - (length/2);
    _settextcolor(BRWHITE);
    _settextposition(1.begin);
    _outtext(cont_msg);
```

```
    /* free memory */
    free_entity_list();
    free_relship_list();
    return;

}

/*********************************************************

        BUILD_CALLING_SEQUENCE

    function to build the calling sequence for genus ename
by looking at the relship linked list

*********************************************************/

void build_calling_seq(ename,calling_seq)

char    ename[20],          /* name of entity we are buiding
                            calling sequence for */
        calling_seq[80];    /* buffer to concatenate
                            calling seq */
{

    int  length;            /* length of calling seq */
    Relship   *rp;


    rp = rhead;

    strcpy(calling_seq,"(");

    while(rp != NULL)   {

        if(strcmp(rp->e1name,ename) ==0)   {

            strcat(calling_seq,rp->e2name);
            strcat(calling_seq,",");
        }
        rp = rp->next;
    }

    /* add right paren at end in place of final comma */

    length = strlen(calling_seq);
    calling_seq[length-1] = ')';

    return;

}
```

```
/*************************************************************

            EDIT_SCHEMA

   function to allow user to add the index statement, the
   generic range and the natural language interpretation
   that were not generated by the parser

*************************************************************/

char * mline1 = "ADD MODEL DOCUMENTATION";
char * mline2 = "Enter (E)dit (S)kip or (Q)uit";


void edit_schema()

{

   Entity     * ep;          /* temp pointer to entity
                             linked list */

   int  length,              /* length of centered msg */
        done = FALSE,        /* finished flag */
        begin,               /* begining col of centered
                              text */
        line,                /* current text line */
        c;                   /* value of character
                             returned from getche */

   char string[2],           /* buffer for edit choice */
   line_buf[81];        /* buffer for mnemonic name */


   read_schema();

   /* initialize full screen text window */

   _setbkcolor(BLACK);
   _settextwindow(1,1,25,80);
   _clearscreen(_GWINDOW);

   length = strlen(mline1);
   begin = 40 - (length/2);
   _settextcolor(RED);
   _settextposition(4,begin);
   _outtext(mline1);
   _settextposition(8,25);
   _settextcolor(BRWHITE);
   _outtext(mline2);
   _setcolor(RED);
   box(3,begin-1,1,length);
```

```
ep = ehead;


while(ep != NULL)    {

     while ( ! done)      {
          _settextcolor(BRWHITE);

          _settextposition(12,5);
          _outtext("Genus Name:  ");

          _outtext(ep->ename);
          _settextposition(12,42);
          _outtext("Index:  ");
          _outtext(ep->etype);


          /* check to see what we want to do */


          fflush(stdin);
          c = getch(stdin);


          switch(c) {

          case 'e':
          case 'E':
               {
                    /* set new text window for prompts */

                    _setbkcolor(BLACK);
                    _settextwindow(14,1,25,80);
                    _clearscreen(_GWINDOW);

               /* edit idx_stmt grange and comments */
               fflush(stdin);
               _settextposition(2,5);
               _outtext("Enter the Index Statement:"  );
               _settextposition(3,5);
               gets(line_buf);
               strcpy(ep->idx_stmt,line_buf);
               _settextposition(5,5);
               _outtext("Enter the Generic Range");
               _settextposition(6,5);
               gets(line_buf);
               strcpy(ep->grange,line_buf);
               _settextposition(8,5);
               _outtext("Enter the Interpretation:" );
               _settextposition(9,5);
               gets(line_buf);
```

```
                    strcpy(ep->comments,line_buf);
                    done = TRUE;

                         /* clear the prompts */
                    _clearscreen(_GWINDOW);
                    /* text window is full screen */
                    _settextwindow(1,1,25,80);

                         break;
                    }
               case 's':
               case 'S':
                    {
                         /* skip this one */
                         done = TRUE;
                         break;
                    }
               case 'q':
               case 'Q':
                    {
                         /* save changes and free memory */
                         delete_from_entity();
                         write_entity();
                         free_entity_list();
                         return;
                    }
               default:
                    {
                         continue;
                    }

               }

          }
          clear_line(12);
          ep = ep->next;
          done = FALSE;
     }
     /* write to ORACLE and free memory */

     delete_from_entity();
     write_entity();
     free_entity_list();
     return;

}
```

```
/************************************************************

DATE:        22 Jan 89 - dsh

FILE:        enter.c

CONTENTS:    Functions that allow the user to enter a new
model in mathematical format and convert this format to
the entity and relship tables defined in the ORACLE
database.

************************************************************/
#include "defs.h"       /* constant definations */
#include <malloc.h>     /* memory allocation function defs */
#include <stdlib.h>     /* std ansi lib defs */
#include <stdio.h>      /* standard i/o function defs */
#include <graph.h>      /* graphics defs     */
#include "symbol.h"     /* symbol table defs */
#include <bios.h>       /* defs for BIOS call to keyboard */
#include "ytab.h"       /* defs for symbol types */
#include <string.h>     /* defs for strlen function */
#include <ctype.h>      /* defs for character
                        type functions */


/* function prototypes for the functions in the file */

void yyerror(void);
void delete_equation(int);
void translate_table(void);
void clear_line(int);
void install_entity(char *,char *,char *,char *,
                      char *,char *,char *,char *);
void install_relship(char *,char *,char *,char *,char *);
void enter_model_description(void);
void rename_genus_relship(char*,char*);
void free_entity_list(void);
void free_module_list(void);
void free_relship_list(void);
void change_relship_type(char *);
void build_monotone_order(char*);
void install_module(char*,char*,char*,char*,char*,int);
void build_grules(void);
void change_symbol(char*,char*);
void insert_grule(char*,int);
```

```
/* global head and tail for the symbol table linked list */

Symbol *head = NULL;
Symbol *tail = NULL;

/* global head and tail for the Entity table linked list */

Entity * ehead = NULL;
Entity * etail = NULL;

/* global head and tail for the Relship table linked list */


Relship * rhead = NULL;
Relship * rtail = NULL;

/* global head and tail for the Module table linked list */

Module * mhead = NULL;
Module * mtail = NULL;

/* head and tail for the Position table linked list */

extern Position * phead;
extern Position * ptail;

/**********************************************************
                 YYERROR

  Called by the parser in the case of a syntax error.
Opens text window and displays location of the error.
**********************************************************/

void yyerror()

{
  char  buffer[80];              /* buffer to hold text
                                 error message */
  extern char yytext[];          /* text buffer used by
                                 YACC parser to hold current
                                  token */

  long old_color;                /* old background color */
  int  mlength;                  /* length of text message
                                     for centering */

  old_color = _getbkcolor();
```

```
    _settextwindow(15,10,20,70);
    _settextcolor(BRWHITE);
    _setbkcolor(CYAN);
    _clearscreen(_GWINDOW);

    /* output syntax error found */
    sprintf(buffer,
    "Syntax Error-> Unexpected Symbol: %s",yytext);
    mlength = strlen(buffer);
    _settextposition(2,30-(mlength/2));
    _outtext(buffer);

    sprintf(buffer,"Any key to continue......");
    mlength = strlen(buffer);
    _settextposition(6,30-(mlength/2));

    _outtext(buffer);

    /* wait for any key */
    _bios_keybrd(_KEYBRD_READ);

    /* reset screen */

    _setbkcolor(old_color);
    _clearscreen(_GWINDOW);
    _settextwindow(1,1,25,80);
    _settextposition(5,5);

}
/********************************************************
                        ENTER_MODEL

    Function that will prompt the user for model to input.
Calls function to write the model to the ORACLCE database.

**********************************************************/

char omess[] =
"Enter the Objective function or END to exit to main menu";

int lomess = sizeof(omess);        /* length of message used
                                   for centering */
 extern char * input_pointer;      /* pointer to the current
                                   input character to be read
                                   by scanner */
 char  equation_buffer[80];        /* buffer that holds
                                    current  equation */


void enter_model()
```

```
{

  int  equation_no = 0;     /* counter for the equations */
  char buffer[80],          /* buffer for prompts */
  ebuf[3],                  /* buffer for
                            equation_no string */
  name[20],                 /* model name */
  name1[20];                /* input model name */
  int  status = 1;          /* parser error status */
  int  lmess;               /* length of message
                             used for centering */
  int  line;                /* current text line */

  char eq[6];                   /* text string used to
                         concatenate with equation #
                        for a temporary variable name */
  int i;                /* index */

  /* window is full screen */

  _setbkcolor(BLACK);
  _settextcolor(BRWHITE);
  _settextwindow(1,1,25,80);
  _clearscreen(_GWINDOW);

  _settextposition(5,5);
  _outtext("Enter the model Name> ");
  gets(name1);
  clear_line(5);



  strcpy(name,"M_");
  strcat(name,name1);
  install_entity(name,"model"," "," "," "," "," "," ");

  cursor_on();
  line = 5;
  strcpy(eq,"EQN");

  /* enter the objective function */

  while(status == 1)  {

       _settextcolor(BRWHITE);

       _settextposition(3,40-(lomess/2));

       _outtext(omess);

       _settextposition(line,5);
```

```
    _outtext("OBJ> Z = ");

    status = yyparse(equation_no);


    /* check to see if syntax error - if so
            remove symbols from table */

    if (status == 1)    {

         *input_pointer = NULL;

         delete_equation(equation_no);

         clear_line(3);
         clear_line(5);

    }

}

/* install objective function as entity */

for(i = 0;i<80;i++) {
     if(iscntrl(equation_buffer[i]) != 0)     {
          equation_buffer[i] = NULL;
     }
}

if(tail->s_type != END)  {
     install_entity("EQN0","f"," "," "," "," ",
     equation_buffer," ");
}

/* set variable for second equation */
line++;
equation_no++;
status = 1;

/* loop until the END symbol is entered */


while(tail->s_type != END)    {

     clear_line(3);
     clear_line(line);

     sprintf(buffer,
     "Enter Equation for constraint Number
     %d or END after last constraint",  equation_no);

     lmess = strlen(buffer);
```

```
      _settextposition(3,40-(lmess/2));

      _outtext(buffer);

      _settextposition(line,5);

      sprintf(buffer,"EQN %d> ",equation_no);

      _outtext(buffer);

      status = yyparse(equation_no);

/* check to see if syntax error - if so
              remove symbols from table */

      if (status == 1)     {

           *input_pointer = NULL;

           delete_equation(equation_no);

      }
      else {
           /* install constraint as entity */

           itoa(equation_no,ebuf,10);
           strcat(eq,ebuf);

           for(i = 0;i<80;i++) {
                if(iscntrl(equation_buffer[i]) != 0)     {
                     equation_buffer[i] = NULL;
                }
           }

           if (tail->s_type != END) {
                install_entity(eq,"t"," "," "," "," ",
                equation_buffer," ");
           }
           eq[3] = NULL;
           equation_no++;
           line++;
      }

}

translate_table();

enter_model_description();

build_grules();
```

```
  build_monotone_order(name);

  write_entity();

  write_relship();

  write_module();

  free_entity_list();

  free_relship_list();

  free_module_list();

  return;

}

**********************************************************
                 CLEAR_LINE

  Function that will clear text line.  Line is cleared in
  the currently defined text window.

*********************************************************/

void clear_line(line)

int    line;          /* line number to be cleared */

{

  char buffer[80];

  _settextposition(line,1);

  sprintf(buffer,
  "
              ");
  _outtext(buffer);

}
```

```
/**********************************************************
                    DELETE_EQUATION

   Function to remove excess symbols from the end of the
symbol table when a syntax error occurs.  The symbol table
is referenced by the global head and tail pointers.  The
symbols to be deleted will always be the last on the linked
list

**********************************************************/

void delete_equation(eqno)

int     eqno;           /* equation number to be deleted */

{

   Symbol     * lp,      /* temp pointers to
                         symbol table entries */
              * tp;


/* no symbols on list */

   if (head == NULL)    {
        return;
   }

   else {

        /* equation is the only entry on the table */

        if (head->equation == eqno)    {

             /* set pointer to first symbol to delete */

             tp=lp=head;

             /* set head and tail to empty list */

             head = tail = NULL;

             /* free memory used in symbols */

             while(lp != NULL)    {

                  lp = tp->next;
                  free((void *)tp);
                  tp = lp;
             }
```

```
        return;
    }

    /* more than one equation on the symbol table */

    else {

        tp = head;
        lp = tp->next;

        while(lp != NULL)   {

            if(lp->equation == eqno) {
                tail = tp;

                /* free memory used in symbols */

                while(lp != NULL)   {
                    tp = lp;
                    lp = tp->next;
                    free((void *)tp);
                }

                return;
            }

            else {

                tp = lp;
                lp = tp->next;
            }

        }


    }

  }

}
```

```
*******************************************************
            TRANSLATE_TABLE

   Function to translate the symbol table created by the
scanner to the form needed for entry into the relational
tables of ORACLE RDBMS

*******************************************************/

void translate_table()

{

   Symbol      *lp,
               *tp; /* leading and trailing pointers used for
                       walking down the symbol table list */
   int         i,   /* index for counting */
               length;   /* number of indices
                       in compound index */
   char ibuf[2];  /* index buffer for seperating
                  compound indices */

   char ebuf[3];  /* temp buffer to hold equation name */

   char eq[6];         /* text string used to concatenate
                       with equation # for a temporary
                       variable name */


   /* initailize pointers to the begining of list */

   tp = head;


   if (head == NULL)    {
        fprintf(stderr, "Th: symbol table is empty -
        Please reenter your model");
        exit(1);
   }

   lp = head->next;


   strcpy(eq,"EQN");

   /* go through entire table */

   while(lp != NULL)    {

        switch(tp->s_type)   {

        case END:
```

```
                {
                        return;
                }

        case IDENTIFIER:
                {

                        if (lp->s_type == INDEX) {

                                length = strlen(lp->s_name);

                                if (length > 1)     {

install_relship("CALLS",tp->s_name,"a",lp->s_name,"ce");
install_entity(lp->s_name,"ce"," "," "," "," "," " ," ");

                                        /* break up index */

                                        for (i=0;i<length;i++)    {
                                                ibuf[0] = lp->s_name[i];
                                                ibuf[1] = NULL;
install_entity(ibuf,"pe"," ",ibuf," "," "," "," ");

install_relship("CALLS",lp->s_name,"ce",ibuf,"pe");

                                        }


                                }

                                else {

install_relship("CALLS",tp->s_name,"a",lp->s_name,"pe");
install_entity(lp->s_name,"pe"," ",lp->s_name," "," "," ","
                                                ");


                                }

install_entity(tp->s_name,"a"," "," "," "," "," "," ");

                                itoa(tp->equation,ebuf,10);
                                strcat(eq,ebuf);
                                if(tp->equation == 0)    {
                                        install_relship("CALLS",eq,"f",
                                        tp->s_name,"a");
                                }
                                else {
                                        install_relship("CALLS",eq,"t",
                                        tp->s_name,"a");
                                }
```

```
                        eq[3] = NULL;
                        tp = lp;
                        lp = tp->next;
                        break;

                    }
                }
            default:
                {

                    tp = lp;
                    lp = lp->next;
                    break;
                }

            }
        }

        return;

}

********************************************************
                 INSTALL_ENTITY

   function to install entity found in LP model into
   entity  linked list.  Creates list if list is empty and
   checks to aovid double entries

********************************************************/

void install_entity(ename,etype,dname,index,
                 index_stmt,grange,grule,comments)


char    ename[20];      /* name of entity  */
char    etype[4];       /* type of entity */
char    dname[30];      /* descriptive name of entity */
char    index[8];       /* index set */
char    index_stmt[50];     /* index statement */
char    grange[20];         /* generic range stmt */
char    grule[80];          /* generic rule */
char    comments[80];       /* informal interpertation */

{

   Entity     * ep,     /* temp pointer to structure */
              *tp;      /* temp pointer to check
                        for duplicates */

   int  same;           /* return variable from
                        strcmp() 0 if entity
```

```
                                   is in list */

    /* dynamically allocate memory for structure */

    ep = (Entity *) malloc(sizeof(Entity));

    /* assign values to structure */

    strcpy(ep->ename,ename);
    strcpy(ep->etype,etype);
    strcpy(ep->dname,dname);
    strcpy(ep->idx,index);
    strcpy(ep->idx_stmt,index_stmt);
    strcpy(ep->grange,grange);
    strcpy(ep->grule,grule);
    strcpy(ep->comments,comments);
    ep->next = NULL;


    /* is list empty - add to head */

    if (ehead == NULL)   {

         ehead = etail = ep;
         return;

    }
    /* check list for duplicates */

    tp = ehead;

    while (tp != NULL)   {

         same = strcmp(ename,tp->ename);

         if (same == 0) {

              /* already in list */
              free(ep);
              return;
         }
         else

         tp = tp->next;
    }
    /* not found - so add to end of list */
    etail->next = ep;
    etail = ep;
    return;

}
```

```
/********************************************************

            INSTALL_RELSHIP

   installs relationship found in LP model into relship
table. checks to see if table is empty or if the
relationship is already represented.

*********************************************************/

void install_relship(rtype,e1name,e1type,e2name,e2type)


char    rtype[12];          /* type of relationship */
char    e1name[20];         /* name of calling element */
char    e1type[8];          /* type of calling element */
char    e2name[20];         /* name of called element */
char    e2type[8];          /* type of cleed element */


{

   Relship    * rp,         /* temp pointer to structure */
              *tp;          /* temp pointer to check
                            for duplicates */

   int  same1,same2;        /* return variables from
                             strcmp() 0 if entity
                            is in list */

   /* dynamically allocate memory for structure */

   rp = (Relship *) malloc(sizeof(Relship));

   /* assign values to structure */

   strcpy(rp->rtype,rtype);
   strcpy(rp->e1name,e1name);
   strcpy(rp->e1type,e1type);
   strcpy(rp->e2name,e2name);
   strcpy(rp->e2type,e2type);
   rp->next = NULL;


   /* is list empty - add to head */

   if (rhead == NULL)  {

        rhead = rtail = rp;
        return;

   }
```

```
    /* check list for duplicates */

    tp = rhead;

    while (tp != NULL)  {

          same1 = strcmp(e1name,tp->e1name);
          same2 = strcmp(e2name,tp->e2name);

          if ((same1 == 0) && (same2 == 0))  {


                /* already in list */

                free(rp);
                return;
          }

          else

          tp = tp->next;

    }

    /* not found - so add to end of list */

    rtail->next = rp;
    rtail = rp;
    return;

}


/*******************************************************

            ENTER_MODEL_DESCRIPTION

    function to enter the mnemonic genus names into the
model description.

*******************************************************/

char * line1 = "ENTER MODEL DESCRIPTIVE INFORMATION";
char * line2 = "Genus Name_____Genus Type_____Mnemonic Genus
Name_____Generic Rule";
char * line3 = "Writing to ORACLE RDBMS table .....";

void enter_model_description()

{

    Entity    * ep;     /* temp pointer to entity
```

```
                    linked list */
int  length,        /* length of centered msg */
begin,              /* begining col of centered text */
line;               /* current text line */

char name[20],      /* buffer for mnemonic name */
yorn[2];            /* buffer for Yes or No answer */

/* initialize text window */

_setbkcolor(BLACK);
_settextcolor(BRWHITE);
_settextwindow(1,1,25,80);
_clearscreen(_GWINDOW);
cursor_on();


length = strlen(line1);
begin = 40 - (length/2);
_settextcolor(RED);
_settextposition(4,begin);
_outtext(line1);
_settextcolor(BRWHITE);
_settextposition(6,1);
_outtext(line2);

box(3,begin-1,1,length);

line = 7;
ep = ehead;

/* put out genus names */

while(ep != NULL)   {

     _settextposition(line,5);
     _outtext(ep->ename);
     _settextposition(line,22);
     _outtext(ep->etype);
     _settextposition(line,47);
     _outtext(ep->grule);

     line++;
     ep = ep->next;

}

line = 7;
ep = ehead;

/* accept mnemonics */
```

```
    while(ep != NULL)    {
         _settextposition(line,30);
         gets(name);

         /* check to see if this is a decision variable */

         if (strcmp(ep->etype,"a") == 0)     {
              _settextposition(line,45);
              _outtext("Decision variable ?(Y/N)   ");
              gets(yorn);
              if (strcmpi(yorn,"y") == 0)    {
                   strcpy(ep->etype,"va");
                   change_relship_type(ep->ename);
              }

         }

         /* check to see if new name was entered
              and replace if it was */

         if (isalnum(name[0]) != 0)     {

              /*  name for pe will include index */

              if (strcmp(ep->etype,"pe") == 0)    {
                   strcat(name,ep->ename);
              }

              rename_genus_relship(ep->ename,name);
              change_symbol(ep->ename,name);

              /* replace with new descriptive name */
              strcpy(ep->ename,name);
         }

         ep = ep->next;
         line++;
    }

    length = strlen(line3);
    begin = 40 - (length/2);
    _settextcolor(BRWHITE);
    _settextposition(23,begin);
    _outtext(line3);

}
```

```
/*********************************************************

            RENAME_GENUS_RELSHIP

  checks through the entire relship linked list to replace
oldname with the mnemonic new name.

*********************************************************/

void rename_genus_relship(oldname,newname)

char    * oldname,     /* name of genus to be renamed */
        * newname;        /* replacement mnemonic name */
{

  Relship * rp;        /* temp pointer into relship
                       linked list */

  int  test;           /* return value from strcmp = 0
                       if match */

  /* pointer to top of list */

  rp = rhead;

  /* check through all names on the list */

  while (rp != NULL)  {

       /* check calling element */

       test = strcmp(rp->e1name,oldname);

       if(test == 0)  {
            strcpy(rp->e1name,newname);
       }

       /* check called element */

       test = strcmp(rp->e2name,oldname);

       if(test == 0)  {
            strcpy(rp->e2name,newname);
       }

       rp = rp->next;

  }

}
```

```
/*********************************************************
                  CHANGE_RELSHIP_TYPE

  checks through the entire relship linked list to replace
change the entity type of ename to va.  This is caled when a
decision variable is identified

**********************************************************/

void change_relship_type(ename)

char    * ename;        /* name of decison variable */

{

  Relship * rp;         /* temp pointer into relship
                        linked list */


  /* pointer to top of list */

  rp = rhead;

  /* check through all names on the list */

  while (rp != NULL)  {

       /* check calling element */


       if(strcmp(ename,rp->e1name) == 0)   {
            strcpy(rp->e1type,"va");
       }

       /* check called element */

       if(strcmp(ename,rp->e2name) == 0)   {
            strcpy(rp->e2type,"va");
       }

       rp = rp->next;

  }

}
```

```
/*****************************************************

             FREE_ENTITY_LIST

  frees dynamic memory used by entity linked list

*****************************************************/

void free_entity_list()

{

  Entity    *lp,
  *tp;

  tp = ehead;
  lp = tp->next;

  ehead = NULL;

  while (lp != NULL)  {

       free(tp);
       tp = lp;
       lp = tp->next;
  }

  free(tp);

}

/********************************************************

             FREE_RELSHIP_LIST

  frees dynamic memory used by relship linked list

********************************************************/

void free_relship_list()

{

  Relship   *lp,
  *tp;

  tp = rhead;
  lp = tp->next;

  rhead = NULL;

  while (lp != NULL)  {
```

```
        free(tp);
        tp = lp;
        lp = tp->next;
    }

    free(tp);

}

/**********************************************************

                FREE_MODULE_LIST

    frees dynamic memory used by module linked list

**********************************************************/

void free_module_list()

{

    Module    *lp,
    *tp;

    tp = mhead;
    lp = tp->next;

    ehead = NULL;

    while (lp != NULL)  {

        free(tp);
        tp = lp;
        lp = tp->next;
    }

    free(tp);

}
/*********************************************************

                FREE_POSITION_LIST

    frees dynamic memory used by position linked list

*********************************************************/

void free_position_list()

{
```

```
    Position  *lp,
    *tp;

    tp = phead;
    lp = tp->next;

    phead = NULL;

    while (lp != NULL)  {

         free(tp);
         tp = lp;
         lp = tp->next;
    }

    free(tp);

}

/*********************************************************

              BUILD_MONOTONE_ORDER

    provides the modular structure in monotone order -
insuring no forward references. Calls write_modular to enter
into ORACLE relship table

*********************************************************/

void build_monotone_order(name)

char    name[20];       /* model name */

{
    Relship    *rp;       /* pointer to relship linked list */
    Module     *mp;       /* pointer to module linked list */
    int        rel_pos = 1;   /* relative position in
                               the order */



    /* install all pe on level 1 */

    rp = rhead;


    while (rp != NULL)  {

         if (strcmp(rp->e2type,"pe") == 0)  {
              install_module("CONTAINS",name,"model",
              rp->e2name,rp->e2type,rel_pos);
```

```
            rel_pos++;
        }

        rp = rp->next;
    }

    /* initialize pointers to begining of lists */

    rp = rhead;
    mp = mhead;


    while (mp != NULL)   {

        /* check for all elements that call those on the
            list */

        while(rp != NULL)    {

            if (strcmp(mp->e2name,rp->e2name) == 0) {

                install_module("CONTAINS",name,"model",
                rp->e1name,rp->e1type,rel_pos);
                rel_pos++;
            }
            rp = rp->next;

        }

        mp = mp->next;
        rp = rhead;

    }


}


/************************************************************

                INSTALL_MODULE

  This function places each module found in into the
module linked list. If the element is already on the
list in the same display level it will not be added.  If it
is already on the list on a lower display level the lower
level is converted to a place holder to allow the arc to
span levels
**********************************************************/

void
install_module(rtype,e1name,e1type,e2name,e2type,rel_pos)
```

```
char    rtype[12];          /* type of relationship */
char    e1name[20];         /* name of calling element */
char    e1type[12];         /* type of calling element */
char    e2name[20];         /* name of called element */
char    e2type[12];         /* type of called element */
int     rel_pos;       /* relative position in the order */
{

  Module          *mp; /* temp pointer to new structure */
  Module          *tp;    /* temp pointer to linked list
                             used to check if element is
                              already in list */


  /* create new structure */

  mp = (Module *) malloc (sizeof(Module));

  strcpy(mp->rtype,rtype);
  strcpy(mp->e1name,e1name);
  strcpy(mp->e1type,e1type);
  strcpy(mp->e2name,e2name);
  strcpy(mp->e2type,e2type);
  mp->rel_pos = rel_pos;
  mp->next = NULL;


  /* is list empty - add to head */

  if (mhead == NULL)  {

       mhead = mtail = mp;
       return;

  }

  /* check list for to see if the element is there already
       if it is there is a forward reference and we need to

       change the rel_pos of the element on the list */

  tp = mhead;

  while (tp != NULL)  {


       if (strcmp(e2name,tp->e2name) == 0)     {
```

```
                /* already in list - change the rel_pos to
                 eliminate forward reference */

                tp->rel_pos = rel_pos;
                free(mp);
                return;
           }

           else

           tp = tp->next;

      }

      /* not found - so add to end of list */

      mtail->next = mp;
      mtail = mp;
      return;

}

/**********************************************************
            CHANGE_SYMBOL

      This function will change the mathematical symbol in
      the symbol table to the new descriptive name entered.
      This will allow us to construct a generic rule using
      the descriptive names

 **********************************************************/

void change_symbol(oldname,newname)

char      oldname[20],      /* mathematical symbol name */
          newname[20];      /* new descriptive name */

{

      Symbol      * sp;      /* temp pointer to symbol
                             table list */

      sp = head;

      while (sp != NULL)  {

           if (sp->s_type == IDENTIFIER) {

                if(strcmp(sp->s_name,oldname) == 0)      {
                     strcpy(sp->s_name,newname);
                }
           }
```

```
        sp = sp->next;
    }

}

/**********************************************************

               BUILD_GRULES

    This function will concatenate the generic rule
    from the symbol table after the descriptive names
    have been entered.

******** ************************************************/

void build_grules()

{

    Symbol    *sp;      /* temp pointer to symbol table */
    int  equation = 0;  /* counter for the equation number */

    char grule_buf[80]; /* temp buffer for grule */

    sp = head;

    grule_buf[0] = NULL;

    while(sp->s_type != END) {

         if (sp->equation == equation) {
              strcat(grule_buf,sp->s_name);
         }
         else {
              insert_grule(grule_buf,equation);
              grule_buf[0] = NULL;
              strcat(grule_buf,sp->s_name);
              equation ++;
         }

         sp = sp->next;
    }
    insert_grule(grule_buf,equation);

}
```

```
/************************************************************

            INSERT_GRULE

  Function to insert the new descriptive generic rule
  into the entity linked list.  The rule is inserted
  into the function of test element that corresponds
  to the number equation.  Note: This function relies
  on the fact that the the f and t entities are inserted
  in the order the are entered as mathematical equations.

************************************************************/


void insert_grule(grule,eq_no)

char    grule[80];      /* descriptive grule to enter */
int     eq_no;          /* corresponding equation number */

{

  Entity    *ep;        /* temp pointer to entity linked list
*/

  int  count =0, /* count of the f or t entity found */
  ftest,         /* return variables from strcmp test */
  ttest;

  ep = ehead;

  while(ep != NULL)   {

       ttest = strcmp(ep->etype,"t");
       ftest = strcmp(ep->etype,"f");

       if (ftest == 0 || ttest == 0) {

            if (count == eq_no) {
                 strcpy(ep->grule,grule);
                 return;
            }
            count++;

       }

       ep = ep->next;
  }

}
```

```
/*******************************************************
DATE:         1 Jan 89 - dsh

FILE:         oracle_r.pc

CONTENTS:     function to read from the ORACLE database.
This is the C version.  This file must be precompiled with
the PRO*C precompiler before compliling with the other
files.  This precompilitation is automatically done by using
MAKE.
*******************************************************/

#include "symbol.h"
#include <stdio.h>


/* fuction prototypes */

int read_entity(void);
int read_relship(void);
void delete_from_relship(void);
void delete_from_entity(void);


extern Entity     * ehead;
extern Relship    * rhead;


EXEC SQL BEGIN DECLARE SECTION;

VARCHAR      uid[20];
VARCHAR      pwd[20];
VARCHAR      ename[20];
VARCHAR      etype[12];
VARCHAR      dname[20];
VARCHAR      idx[4];
VARCHAR      idx_stmt[50];
VARCHAR      grange[20];
VARCHAR      grule[80];
VARCHAR      comments[80];

VARCHAR      rtype[8];
VARCHAR      e1name[20];
VARCHAR      e1type[12];
VARCHAR      e2name[20];
VARCHAR      e2type[12];

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;
```

```
/************************************************************

             READ_SCHEMA

   Function to read the global entity  linked
list from the corresponding ORACLE RDBMS tables.  Assumes
predefined table and    user: dsh  password: thesis

***********************************************************/


int read_schema()

{
/* login to oracle */


strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);


EXEC SQL WHENEVER ERROR GOTO errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

EXEC SQL DECLARE C1 CURSOR FOR
   SELECT
ENAME,ETYPE,DNAME,IDX,IDX_STMT,GRANGE,GRULE,COMMENTS
   FROM ENTITY,CONTAINZ WHERE ENTITY.ENAME = CONTAINZ.E2NAME
   ORDER BY REL_POS;


EXEC SQL OPEN C1;


EXEC SQL WHENEVER NOT FOUND GOTO finish;

for(;;) {

   EXEC SQL FETCH C1 INTO
:ename,:etype,:dname,:idx,:idx_stmt,
   :grange,:grule,:comments;

   /* string is returned without so
        NULL we need to add it */

   ename.arr[ename.len] = NULL;
   etype.arr[etype.len] = NULL;
   dname.arr[dname.len] = NULL;
   idx.arr[idx.len] = NULL;
```

```
   idx_stmt.arr[idx_stmt.len] = NULL;
   grange.arr[grange.len] = NULL;
   grule.arr[grule.len] = NULL;
   comments.arr[comments.len] = NULL;

   install_entity(ename.arr,etype.arr,dname.arr,idx.arr,
        idx_stmt.arr,grange.arr,grule.arr,comments.arr);


   }

finish:
EXEC SQL CLOSE C1;

EXEC SQL WHENEVER SQLERROR CONTINUE;

EXEC SQL COMMIT WORK RELEASE;

return;

errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;

}


/************************************************************


            READ_RELSHIP

   Function to read the global relship  linked
list from the corresponding ORACLE RDBMS tables.  Assumes
predefined table and    user: dsh  password: thesis

*********************************************************/

int read_relship()

{

/* login to oracle */


strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER SQLERROR goto errpt;
```

```
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

/* this could be ordered by rel_pos ?? */

EXEC SQL DECLARE C2 CURSOR FOR
   SELECT E1NAME,E1TYPE,E2NAME,E2TYPE
   FROM CALLS ORDER BY E2NAME,E1NAME;

EXEC SQL OPEN C2;

EXEC SQL WHENEVER NOT FOUND GOTO finish;

for (;;)     {

   EXEC SQL FETCH C2 INTO :e1name,:e1type,:e2name,:e2type;

   /* string is returned without so NULL we need to add it
*/

   e1name.arr[e1name.len] = NULL;
   e1type.arr[e1type.len] = NULL;
   e2name.arr[e2name.len] = NULL;
   e2type.arr[e2type.len] = NULL;

    install_relship("CALLS",e1name.arr,
                          e1type.arr,e2name.arr,e2type.arr);

   }

finish:
EXEC SQL CLOSE C2;

EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL COMMIT WORK RELEASE;

return;

errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;



}
```

```
/***********************************************************

             DELETE_FROM_RELSHIP

   Function to delete the ORACLE RDBMS relship table.
Assumes predefined table and    user: dsh  password: thesis

**********************************************************/

void delete_from_relship()

{

/* login to oracle */


strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER SQLERROR goto errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

EXEC SQL DELETE FROM RELSHIP;


finish:
EXEC SQL WHENEVER SQLERROR CONTINUF;

EXEC SQL COMMIT WORK RELEASE;

return;

errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;


}
```

```
/************************************************************
              DELETE_FROM_ENTITY

   Function to delete the ORACLE RDBMS entity table.
Assumes predefined table and    user: dsh  password: thesis

************************************************************/
void delete_from_entity()

{

/* login to oracle */

strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER SQLERROR goto errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

EXEC SQL DELETE FROM ENTITY;


finish:
EXEC SQL WHENEVER SQLERROR CONTINUE;

EXEC SQL COMMIT WORK RELEASE;


return;


errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;


}
```

```
/**********************************************************
DATE:         10 Jan 89 - dsh

FILE:         oracle_w.pc

CONTENTS:     function to write to the ORACLE database.  This
isthe C version.  This file must be precompiled with the
PRO*C precompiler before compliling with the other files.
This precompilitation is automatically done by using MAKE.
**********************************************************/
#include "symbol.h"
#include <stdio.H>


/* fuction prototypes */

int write_entity(void);
int write_relship(void);
int write_module(void);

extern Entity     * ehead;
extern Relship    * rhead;
extern Module     * mhead;


EXEC SQL BEGIN DECLARE SECTION;

VARCHAR       uid[20];
VARCHAR       pwd[20];
VARCHAR       ename[20];
VARCHAR       etype[12];
VARCHAR       dname[20];
VARCHAR       idx[4];
VARCHAR       idx_stmt[50];
VARCHAR       grange[20];
VARCHAR       grule[80];
VARCHAR       comments[80];

VARCHAR       rtype[12];
VARCHAR       e1name[20];
VARCHAR       e1type[12];
VARCHAR       e2name[20];
VARCHAR       e2type[12];
int           rel_pos;

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;
```

```
/************************************************************

            WRITE_ENTITY

  Function to load the global entity  linked
list into the corresponding ORACLE RDBMS tables.  Assumes
predefined table and    user: dsh  password: thesis

*********************************************************/


int write_entity()

{

Entity * ep;      /* temp pointer to linked list */

/* login to oracle */


strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER ERROR GOTO errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

ep = ehead;

while (ep != NULL)     {


  strcpy(ename.arr,ep->ename);
  ename.len = strlen(ename.arr);

  strcpy(etype.arr,ep->etype);
  etype.len = strlen(etype.arr);

  strcpy(dname.arr,ep->dname);
  dname.len = strlen(dname.arr);

  strcpy(idx.arr,ep->idx);
  idx.len = strlen(idx.arr);

  strcpy(idx_stmt.arr,ep->idx_stmt);
  idx_stmt.len = strlen(idx_stmt.arr);

  strcpy(grange.arr,ep->grange);
  grange.len = strlen(grange.arr);
```

```
   strcpy(grule.arr,ep->grule);
   grule.len = strlen(grule.arr);

   strcpy(comments.arr,ep->comments);
   comments.len = strlen(comments.arr);

EXEC SQL INSERT INTO ENTITY
 (ename,etype,dname,idx,idx_stmt,grange,grule,comments)
  VALUES(:ename,:etype,:dname,:idx,
            :idx_stmt,:grange,:grule,:comments);

EXEC SQL COMMIT WORK;

ep = ep->next;

}

EXEC SQL COMMIT WORK RELEASE;

return;

errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;


}

/************************************************************


            WRITE_RELSHIP

   Function to load the global relship  linked
list into the corresponding ORACLE RDBMS tables.  Assumes
predefined table and    user: dsh  password: thesis

***********************************************************/


int write_relship()

{

Relship * rp;

/* login to oracle */


strcpy(uid.arr,"dsh");
```

```
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);


EXEC SQL WHENEVER ERROR GOTO errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

rp = rhead;

while (rp != NULL)    {


   strcpy(rtype.arr,rp->rtype);
   rtype.len = strlen(rtype.arr);

   strcpy(e1name.arr,rp->e1name);
   e1name.len = strlen(e1name.arr);

   strcpy(e1type.arr,rp->e1type);
   e1type.len = strlen(e1type.arr);

   strcpy(e2name.arr,rp->e2name);
   e2name.len = strlen(e2name.arr);

   strcpy(e2type.arr,rp->e2type);
   e2type.len = strlen(e2type.arr);


   EXEC SQL INSERT INTO RELSHIP
   (rtype,e1name,e1type,e2name,e2type)
   VALUES(:rtype,:e1name,:e1type,:e2name,:e2type);

   EXEC SQL COMMIT WORK;

   rp = rp->next;

   }

EXEC SQL COMMIT WORK RELEASE;

return;

errpt:
printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
EXEC SQL ROLLBACK WORK RELEASE;
return;

}
```

```
/*************************************************************

            WRITE_MODULE

   Function to load the global module  linked
list into the relship ORACLE RDBMS table.  Assumes
predefined table and    user: dsh  password: thesis

************************************************************/


int write_module()

{

Module * mp;      /* pointer to the module linked list */


/* login to oracle */


strcpy(uid.arr,"dsh");
uid.len = strlen(uid.arr);
strcpy(pwd.arr,"thesis");
pwd.len = strlen(pwd.arr);

EXEC SQL WHENEVER ERROR GOTO errpt;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

mp = mhead;

while (mp != NULL)     {

   strcpy(rtype.arr,mp->rtype);
   rtype.len = strlen(rtype.arr);

   strcpy(e1name.arr,mp->e1name);
   e1name.len = strlen(e1name.arr);

   strcpy(e1type.arr,mp->e1type);
   e1type.len = strlen(e1type.arr);

   strcpy(e2name.arr,mp->e2name);
   e2name.len = strlen(e2name.arr);

   strcpy(e2type.arr,mp->e2type);
   e2type.len = strlen(e2type.arr);

   rel_pos = mp->rel_pos;
```

```
    EXEC SQL INSERT INTO RELSHIP
    (rtype,e1name,e1type,e2name,e2type,rel_pos)
    VALUES(:rtype,:e1name,:e1type,:e2name,:e2type,:rel_pos);

    EXEC SQL COMMIT WORK;

    mp = mp->next;

    }

  EXEC SQL COMMIT WORK RELEASE;

  return;

  errpt:
  printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  return;

  }
```

## LIST OF REFERENCES


1. Sprague, R. A. Jr., "A Framework for the Development of Decision Support Systems," in R. H. Sprague Jr. and H. J. Watson (ed.) Decision Support Systems: Putting Theory into Practice, Prentice-Hall, 1986.

2. IFPS Users manual, Release 9.0, Execucom Systems 1983.

3. Sprague, R. A. Jr., Building Effective Decision Support Systems, Prentice-Hall, 1982.

4. Little, J. D. C., "Models and Managers: The Concept of a Decision Calculus," Management Science, v.16, no. 8, April 1970.

5. Dolk, D. R. and Konsynski,B. R., "Model Management in Organizations," Information and Management, v. 9, no. 1, August 1985.

6. Dolk, D. R., Model Management and Structured Modeling: The Role of an Information Resource Dictionary System, Communications of the ACM, v. 31, no. 6, June 1988.

7. Geoffrion, A. M., "An Introduction to Structured Modeling," Management Science, v. 33, May 1987.

8. Brooke, A. and Kendrick, D., GAMS: A Users Guide, The Scientific Press, 1988.

9. Lenard, M. L., "Representating Models as Data," Journal of Management Information Systems, vol. 4, no. 2, 1986.

10. Geoffrion, A. M., "SML: A Model Definition Language for Structured Modeling," Working Paper No. 360, Graduate School of Management, University of California Los Angeles, May 1988.

11. Markland, R. E. and Sweigart, J. R., Quantitative Methods: Applications to Managerial Decision Making, John Wiley and Sons, 1987.

12. Dolk, D. R., "Automatic Generation of Structured Models from LP Models," Draft paper, Naval Postgraduate School, Monterey, CA, September 1988.

13. Lesk, M. E., "LEX : A Lexical Analyzer Generator," CSTR 39, Bell Laboratories, Murray Hill, NJ.

14. Johnson, S. C., "YACC: Yet another Compiler-Compiler," CSTR 32, Bell Laboratories, Murray Hill, NJ.

15. Aho, A. V. and Ullman,J. D., Principles of Compiler Design, Addison-Wesley, 1977.

16. Wyant, A. Jr., Design and Implementation of Prototype Graphical User Interface for a Model Management System, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March 1988.

## INITIAL DISTRIBUTION LIST

| | | |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 2 |
| 3. | Computer Technology Curriculum Office, Code 37<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 1 |
| 4. | Professor Daniel R. Dolk, Code 54DK<br>Administrative Sciences Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 1 |
| 5. | Professor Gordon H. Bradley, Code 55BZ<br>Operations Research Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 1 |
| 6. | Commandant(G-PTE)<br>U.S. Coast Guard<br>21100 Second Street, SW<br>Washington, DC 20593 | 2 |
| 7. | LT David S. Hill<br>Commandant(G-PIM)<br>U.S. Coast Guard<br>21100 Second Street, SW<br>Washington, DC 20593 | 1 |